

AD-A155 465

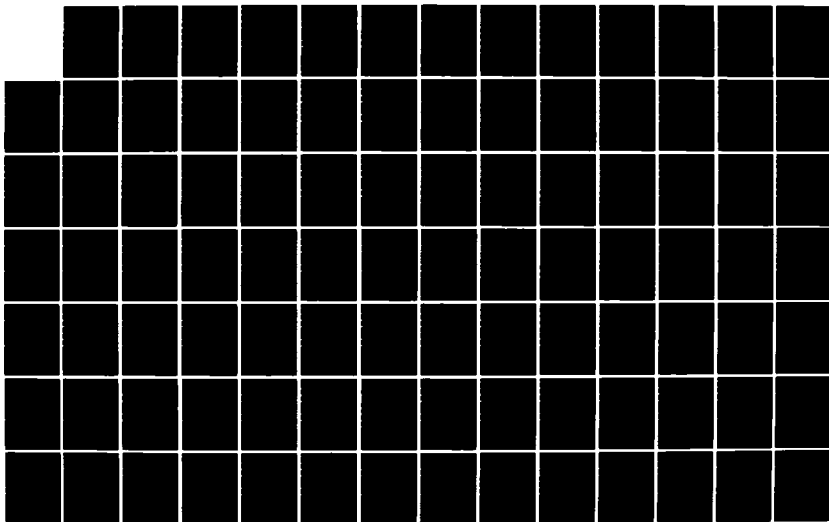
DEVELOPMENT OF A DEDICATED SPEECH WORK STATION(U) AIR  
FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF  
ENGINEERING W H LIEBER DEC 84 AFIT/GE/EE/84D-71

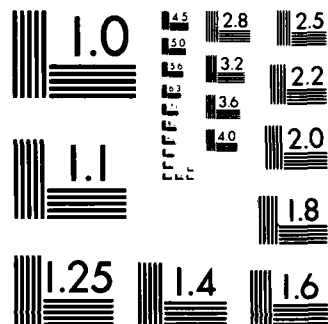
1/3

UNCLASSIFIED

F/G 9/2

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

1

AD-A155 465



DEVELOPMENT OF A DEDICATED  
SPEECH WORK STATION

A

THESIS

AFIT/GE/EE/84D-71

William H. Lieber  
Capt USAF

DTIC FILE COPY

DTIC  
EXCISE  
JUN 20 1985

DEPARTMENT OF THE AIR FORCE  
AIR UNIVERSITY

G

**AIR FORCE INSTITUTE OF TECHNOLOGY**

Wright-Patterson Air Force Base, Ohio

85

5 21 034

AFIT/GE/EE/84D-71

DEVELOPMENT OF A DEDICATED  
SPEECH WORK STATION

THESIS

AFIT/GE/EE/84D-71

William H. Lieber  
Capt USAF

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A/1	



DTIC  
ELECTE  
JUN 20 1985  
S D  
G

Approved for public release; distribution unlimited.

## DEVELOPMENT OF A DEDICATED SPEECH WORK STATION

THESIS

Presented to the Faculty of the School of Engineering  
of the Air Force Institute of Technology  
Air University  
in Partial Fulfillment of the  
Requirement for the Degree of  
Master of Science in Electrical Engineering

by

William H. Lieber, B.S.E.E.  
Capt USAF  
Graduate Electrical Engineering  
December 1984

Approved for public release; distribution unlimited.

## Preface

This thesis project attempted to design and develop a dedicated speech work station. <sup>2.0017</sup> The work station would accept one of sixteen possible analog signal inputs, digitize the input and store the resulting data in the expanded computer memory of a modified Cromemco S-1<sup>00</sup>~~00~~ bus microcomputer system. Once stored, the data could be relocated to magnetic "floppy disk" storage or graphically displayed on a CRT. 50. 8 VI

I would like to thank Major Larry Kizer, my thesis advisor, for his guidance and support during this thesis project. I would also like to thank Major Kenneth G. Castor, my faculty advisor, and Capt Dale Hibner for their encouragement.

Finally, I would like to thank my wife Colletta, my daughter Elizabeth, and my son Matthew for their love and understanding which allowed me the enormous amount of time required to complete this thesis project and graduate from the Air Force Institute of Technology.

William H. Lieber

## Contents

	Page
Preface .....	ii
List of Figures .....	v
Abstract .....	vi
I. Introduction .....	I-1
Background .....	-1
Problem .....	-2
General Approach .....	-2
II. Detailed Analysis .....	II-1
Cromemco/S-100 .....	-1
Hardware Development .....	-4
Software Development .....	-16
Graphics .....	-28
III. Design and Fabrication .....	III-1
Hardware .....	-1
Software .....	-3
IV. Validation .....	IV-1
V. Conclusions and Recommendations .....	V-1
Bibliography .....	BIB-1
Appendix A: Software Program Listings .....	A-1
SPEECH.C .....	-1
SPEECH.H .....	-3
TITLE.C .....	-6
INTRO.C .....	-8
DESCRIBE.C .....	-10
DEFAULTS.C .....	-17
MENU.C .....	-20
QUIT.C .....	-23
ANALOG.C .....	-24
DIGITAL.C .....	-27
STORE.C .....	-30
RETRIEVE.C .....	-34
GRAPHICS.C .....	-38
PLOT.C .....	-44
RIGHT.C .....	-48
LEFT.C .....	-51
VOLTLINE.C .....	-54
INPUT.C .....	-55
DMA.C .....	-61
TIMING.C .....	-65

CLEARMEM.C .....	A-70
NOP.CSM .....	-73
WAIT.CSM .....	-74
Appendix B: Integrated Circuit Layout .....	B-1
BOARD 1 - BOOT UP CONTROL .....	-1
BOARD 2 - DIRECT MEMORY ACCESS .....	-2
BOARD 3 - DIGITAL TO ANALOG CONVERSION ...	-3
BOARD 4 - ANALOG TO DIGITAL CONVERSION ...	-4
Appendix C: List of Integrated Circuits Used ...	C-1
Appendix D: Wiring Diagrams .....	D-1
BOARD 1 - BOOT UP CONTROL .....	-1
BOARD 2 - DIRECT MEMORY ACCESS .....	-2
BOARD 3 - DIGITAL TO ANALOG CONVERSION ...	-5
BOARD 4 - ANALOG TO DIGITAL CONVERSION ...	-6
Appendix E: Timing Diagrams .....	E-1
MEMORY TO MEMORY TRANSFER .....	-1
DIGITAL TO ANALOG CONVERSION .....	-2
ANALOG TO DIGITAL CONVERSION .....	-3
Appendix F: DMA Controller Chip .....	F-1
Appendix G: System Timing Controller Chip .....	G-1
Appendix H: Analog to Digital Module .....	H-1
Appendix I: Digital to Analog Module .....	I-1
Appendix J: MB64 Static RAM Users Guide .....	J-1
Appendix K: Imaginator - Graphics Instruction Set .....	K-1
Appendix L: Adding Machine Language Code to a "C" Language Program .....	L-1
Appendix M: Long (32-bit) Integer Package for a "C" Language Program .....	M-1
Vita: .....	VITA

## List of Figures

Figure		Page
II-1	Analog-to-Digital Conversion .....	II-7
II-2	Digital-to-Analog Conversion .....	II-10
II-3	Data Samples Storage Locations .....	II-12
II-4	Extended Memory Addressing .....	II-13
II-5	Cromemco Boot-up .....	II-14
II-6	Decoder Chip's Port Addresses and Functions .....	II-19
II-7	A-to-D or D-to-A Command Register .....	II-21
II-8	MEM-to-MEM Command Register .....	II-21
II-9	Clear Memory Command Register .....	II-22
II-10	Channel 0 Mode Register .....	II-23
II-11	Channel 1 Mode Register .....	II-23
II-12	Channel 2 Mode Register .....	II-24
II-13	Channel 3 Mode Register .....	II-24
II-14	Master Mode Register .....	II-25
II-15	Out 1 Counter Mode Register .....	II-25
II-16	Out 2 Counter Mode Register .....	II-26
II-17	Out 4 Counter Mode Register .....	II-26
II-18	Out 5 Counter Mode Register .....	II-27
II-19	Graphical Display Format .....	II-29

## Abstract

As a result of the hardware and software developed under this thesis, the AFIT Speech Lab's Cromemco S-1<sup>00</sup>~~00~~ bus microcomputer system can be configured as a dedicated stand alone speech work station.

Hardware is now developed which provides an extended memory capability for storage of analog-to-digital sampled analog speech. Data storage is via a direct memory access (DMA) capability. The hardware also supports providing an analog output from previously stored data samples via a digital-to-analog capability.

Software is developed which controls the analog input to be sampled and the sampling rate to be used. The software also allows the sampled data to be graphically displayed 5~~00~~ samples at a time on a video display screen or to be placed in or returned from more permanent storage on a magnetic disk.

The detailed analysis, development, and fabrication of the hardware and software is also contained in the thesis.

*the thesis describes the fabrication, validation,  
and program development; significant results  
are shown in the figures.*

# DEVELOPMENT OF A DEDICATED SPEECH WORK STATION

## I. Introduction

### Background

The Electrical Engineering department of the Air Force Institute of Technology (AFIT) needed to expand the capability of an existing Cromemco S-100 bus microcomputer system. This system was used to sample and digitally store analog speech signals. The modification was required because the existing 64K system was limited to sampling approximately three seconds of speech. This time restraint reduced the effectiveness of the system as a speech analysis tool. This resulting thesis project involved both the hardware and software modification of the existing system. The hardware modification portion expanded the memory storage capability of the system, thus allowing increased digital storage of sampled analog signals such as speech. The software additions controlled the data sampling, data storage, and graphical display of the digitized analog signals. Chapter One discusses the background of the thesis and its general approach. Chapter Two provides insight into the hardware and software development. Chapter Three addresses the design and fabrication involved in the thesis. And finally, Chapters Four and Five address validation of the thesis effort and provide some conclusions and recommendations.

## Problem

The problem addressed by this thesis is the development of a stand alone speech acquisition and graphical display station. This computer station is used to sample analog speech, provide digital storage of the samples, and allow graphical display of selected portions of the stored data. The speech station is a tool to be used by speech researchers.

## General Approach

First, the dedicated speech station uses the existing equipment located in building 640, room 241. The existing Cromemco S-100 bus microcomputer system and associated video display terminal had to be modified to acquire and display analog speech. The original Cromemco memory is limited to 64K bytes, where 1K equals 1024 bytes and one byte equals eight bits. The memory had to be increased by adding six MB64 64K static Random Access Memory (RAM) boards to the S-100 bus.

Second, a direct memory access (DMA) controller has been added to the microcomputer system. The DMA capability and additional memory are required to allow high sampling rates and to avoid losing any data samples due to the relative long amount of time required by the computer to write the data samples onto magnetic disk for storage. The additional memory is addressed by using an extended memory addressing format.

Third, the video display unit terminal associated with the Cromenco system contains a high resolution graphics capability called Imaginator. This capability allows the system user to acquire, plot and edit selected portions of the data samples stored in the static memory. Data sample manipulation to and from magnetic disk storage is also allowed.

Finally, the "C" programming language is used whenever possible to write the software routines.

## II. Detailed Analysis

This chapter briefly describes the Cromenco S-100 bus microcomputer system. It also includes a discussion of the hardware and software designs necessary for the speech station to operate. Finally, the graphics software is discussed separately.

### Cromenco/S-100

The Cromemco S-100 bus microcomputer consists of a chassis housing a built-in power supply and an S-100 bus mother-board made up of twenty-one 100-pin connectors. Each of these connectors provides a slot into which a S-100 compatible circuit card can be inserted. Each of the slots on the Cromemco mother-board receives the same set of signals; therefore, any circuit card that is compatible with the S-100 bus can be plugged into any of the slots. Consequently, there are no special or reserved slots in the Cromemco system. The only consideration that must be made is that no more than one card can be assigned to the same memory addresses and/or input/output (I/O) address. The built-in power supply provides three unregulated voltages (+8, -18, and +18 volts) to the bus. Each circuit card inserted into a slot has to have its own on-board regulator(s) capable of supplying the power required by the circuit.

The S-100 bus has become one of the most popular hobby

busses and is now considered an industry standard. MITS Inc. first introduced the S-100 bus on their 8080-based Altair computer. Since the S-100 bus is originally designed to support the 8080 processor, most of the bus signals are representative of the signals generated by an 8080 processor. Almost all of the 100 lines of the bus have a standard predefined function and are described in many easily available publications (Ref 1). Basically, the lines are divided into four major groups. These groups consist of the power and ground lines, the address lines, the data lines, and the control lines.

The power and ground line group provides the voltages required by the computer. Six lines are assigned to provide the three unregulated voltages (+8, -18 and +18 volts) and their ground returns. The +8 volts and ground values each appear on two separate lines. The address line group initially provided the sixteen lines required to supply a 16-bit memory address. However, the latest Institute of Electrical and Electronics Engineers (IEEE) 696 standard supports extended 24-bit memory addressing. The data line group is composed of sixteen lines and is used to supply program instructions and data. Normally, the data group lines are separated into two sub-groups of eight lines each. One set is used to supply data to the central processing unit (CPU), while the other set is used for data emanating from the CPU. The IEEE 696 standard allows these sixteen lines to become bidirectional when the proper control signals are provided. This allows the bus to be used with

the evolving 16-bit microprocessors. The IEEE 696 effort is heralded as a standard and should serve to standardize the bus across the industry for both 8-bit and 16-bit operation. The remaining lines comprise the control line group. These lines are used to carry timing and control signals between the CPU, memory, I/O, and any other circuits located on the S-100 bus.

With the acceptance of the S-100 bus, made evident by the extent of its use, it is a good choice as the computer bus for use in this project. The best system, however, is one that uses a 16-bit processor. But since the ground rules are to use the existing hardware as much as possible, the use of a 16-bit processor is not future addressed. Returning to the S-100 bus, there are numerous manufacturers building cards that comply with the standard, making the task of finding a commercially built card, to do all but the most specialized functions, quite easy. Such is the case of using the commercially available MB64 memory card.

Since the S-100 bus is a true parallel bus, all signals within the system are available to any card placed on the bus. This provides for nearly unlimited use of these signals by any circuit card inserted on the S-100 bus. The Cromemco's S-100 bus is capable of communicating directly with 256 I/O ports using the low byte of its address bus. The S-100 bus has two control lines that delineate I/O or control operations from normal memory transactions. One of these lines, SINP, indicates an input operation while the other, SOUT, indicates an output operation. Each interface

must decode the least significant byte of the 16-bit address bus to determine its port addresses when either of these two signals appear. Two other control signals, pWR and pDBIN, are always generated during memory write and read operations, respectively. These signals are used to tell the interface circuit when it must either supply data to or take data from the system.

More detailed information on the Cromemco system can be found in the Cromemco instruction manuals and technical manuals (Refs 2 thru 7).

#### Hardware Development

The following discussion is a brief synopsis of the major hardware components used during the thesis project.

The DMA Controller. The AM9517A Multimode Direct Memory Access (DMA) Controller is a peripheral interface circuit for microprocessor systems. It is designed to allow external devices to directly transfer information to or from the system memory. Memory-to-memory transfer capability is also provided. Both of these capabilities are required and used to support the thesis. The direct transfer of data to the system memory occurred during analog-to-digital sampling of the speech signal. The direct transfer of data from the system memory occurred during the digital-to-analog conversion of the sampled data. The memory-to-memory transfer capability is used to move data samples between the

higher memory addresses (above 64K) and the lower memory addresses (below 64K). This is necessary in order to use existing software commands to move data to and from magnetic disk and during the graphical representation portion of the project. These commands only work on data samples located in the memory addresses located below 64K. More specific details on the DMA module are contained in Appendix F.

The System Timing Controller (STC). The AM9513 STC is a large scale integrated circuit designed to service many types of counting, sequencing and timing applications. This chip is software programmed by the system user of the SPEECH program. The STC OUT5 pin provides the clocking necessary to sample the analog input at the system user specified sampling rate. The output of the OUT4 pin is used to disable the digital-to-analog output until the specific starting sample is reached. The basic digital-to-analog concept is that all stored data samples on a memory board are addressed during each digital-to-analog operation, but only when OUT4 is active high will any external output be allowed. This is required due to the technique used to address the data storage locations in static memory. And finally, the OUT2 pin is used to disable the sampling conversion process. Be it either analog-to-digital or digital-to-analog conversion, the OUT2 pin causes an interrupt signal to be sent to the central processing unit. The interrupt signal causes the conversion process which is running to stop. More specific details on the STC chip are contained in Appendix G.

The A-to-D Module. The DAS 1128 is a complete self-contained miniature high speed data acquisition system. It contains an analog input signal multiplexer, a sample-and-hold amplifier, a 12 bit analog-to-digital converter, and all of the programming, timing, and control circuitry needed to perform the complex data acquisition function. More specific details on the analog-to-digital module are contained in Appendix H. The basic concept for analog-to-digital conversion in the SPEECH program is shown in Figure II-1. The analog-to-digital module is the center of the operation. An analog input is placed at the INPUT of the module. The module is then triggered to sample the analog input. The trigger, however, does not occur unless: (1) the clock sampling pulse is positive; thus assuring the correct number of samples per second will be taken, (2) the analog-to-digital on/off flip-flop is set to positive; thus assuring only an analog-to-digital conversion can take place, and (3) the finish count is positive; thus assuring that the last user requested sample has not yet been taken. Each data sample is placed into static memory via the data bus. It takes two 8-bit bytes to completely store the sampled input -- sampled to 12 bit accuracy. The signal on the analog-to-digital module's end of conversion (EOC) pin is used to toggle between processing the first byte of a sample or the second byte of a sample. The control is via a set of flip-flops. The flip-flops are enabled by the byte count (first or second byte of a sample) and the Address Enable (AEN) line of the DMA module. The Data REQuest 2

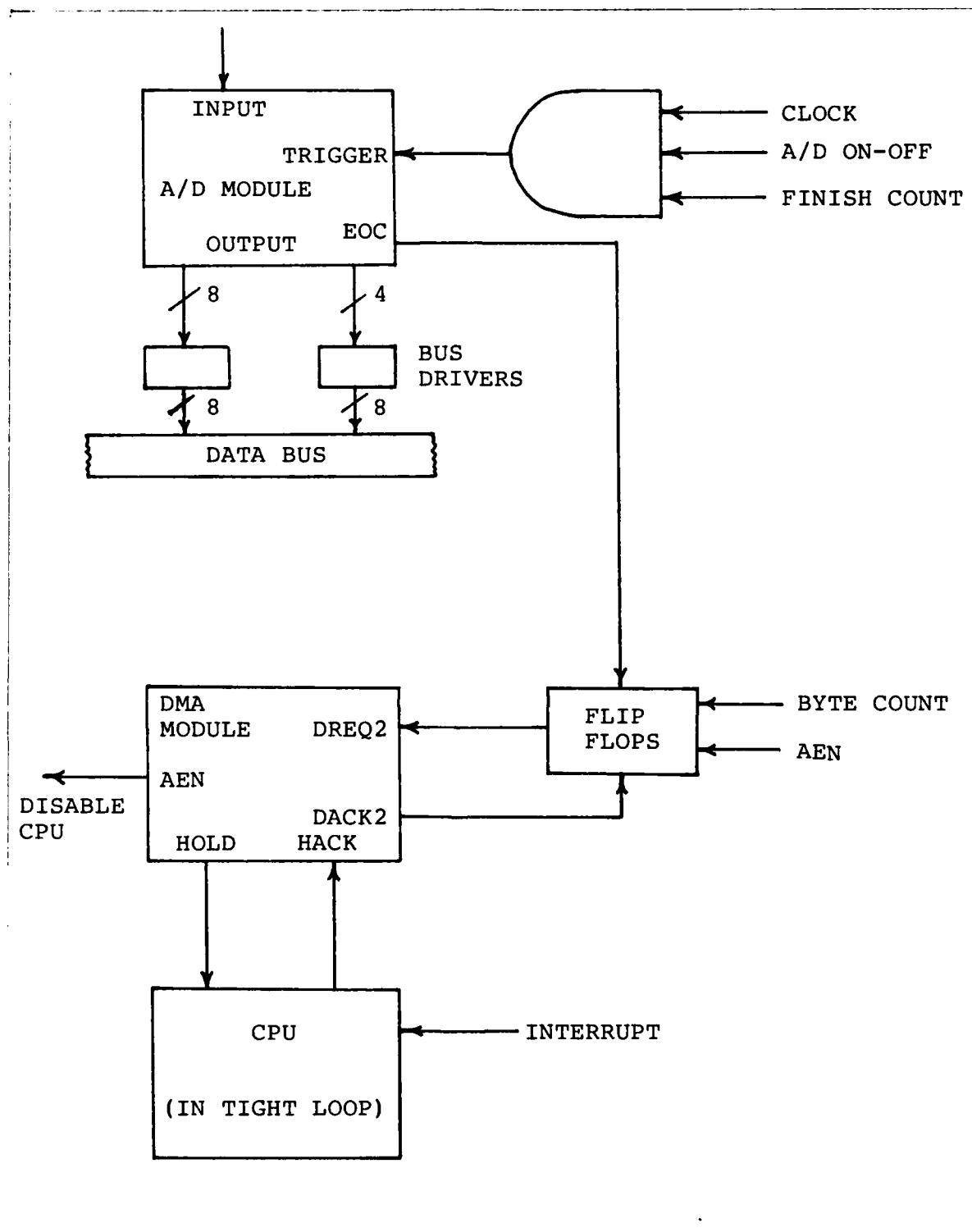


FIG II-1. ANALOG-TO-DIGITAL CONVERSION

(DREQ2) and Data ACKnowledge 2 (DACK2) lines of the DMA controller are used to request and aid control of analog-to-digital sampling. The DREQ2 is used to obtain DMA service. The HOLD and HACK lines are then used by the DMA chip to request the central processing unit to relinquish control of the system buses. The signal on the AEN line of the DMA module is also used to disable the CPU so that only the DMA is in charge of the S-100 bus. Note that just before the DMA module is placed in charge of the data bus, via the HOLD and HACK signal lines, the CPU is placed into a tight loop of doing nothing but no-operation (NOP) commands. The CPU stays in this condition until the CPU's interrupt line is made active. This occurs when the user requested number of data samples has been taken. The system user is then returned to the main SPEECH menu.

The D-to-A Module. The DAC 1118 is a 12 bit, general purpose digital-to-analog converter which comes complete with an input storage register and output amplifier. For this thesis project, the chip is wired to provide a two's complement code output. In addition, the chip is set to provide a bipolar output in the range of plus five volts to minus five volts. More specific details on the digital-to-analog module are contained in Appendix I. The basic concept for digital-to-analog conversion in the SPEECH program is shown in Figure II-2. The digital-to-analog module is the center of the operation. An analog output

from the digital-to-analog module is the result of its' operation. The conversion of a new 12 bit data sample into an analog output is controlled by the STROBE input line. This line is true only when: (1) the sample by-pass signal is true -- meaning all the samples which must be passed over to get to the first sample requested by the system user have been passed over, (2) the output clock pulse is positive; thus assuring the correct number of samples per second will be sent out, (3) the digital-to-analog on/off flip-flop is set to positive; thus assuring only a digital-to-analog conversion can take place, and (4) the finish count signal is positive; thus assuring that the last user requested sample has not yet been sent out. Each data sample is retrieved from static memory via the data bus. It takes two 8-bit bytes to completely send out the correct sampled input -- sampled to 12 bit accuracy. The flip-flop control circuitry toggles between the first byte of a sample and the second byte of a sample. The flip-flops are enabled by the byte count (first or second byte of a sample) and the signal on the AEN line of the DMA module. The Data REQuest 3 (DREQ3) and Data ACKnowledge 3 (DACK3) lines of the DMA controller are used to request the digital-to-analog conversion. The signal on the AEN line of the DMA module is also used to disable the CPU so that only the DMA is in charge of the S-100 bus. Note that just before the DMA is placed in charge of the data bus, via the HOLD and HACK lines, the CPU is placed in a tight loop of doing nothing but no-operation (NOP) commands. The CPU stays in

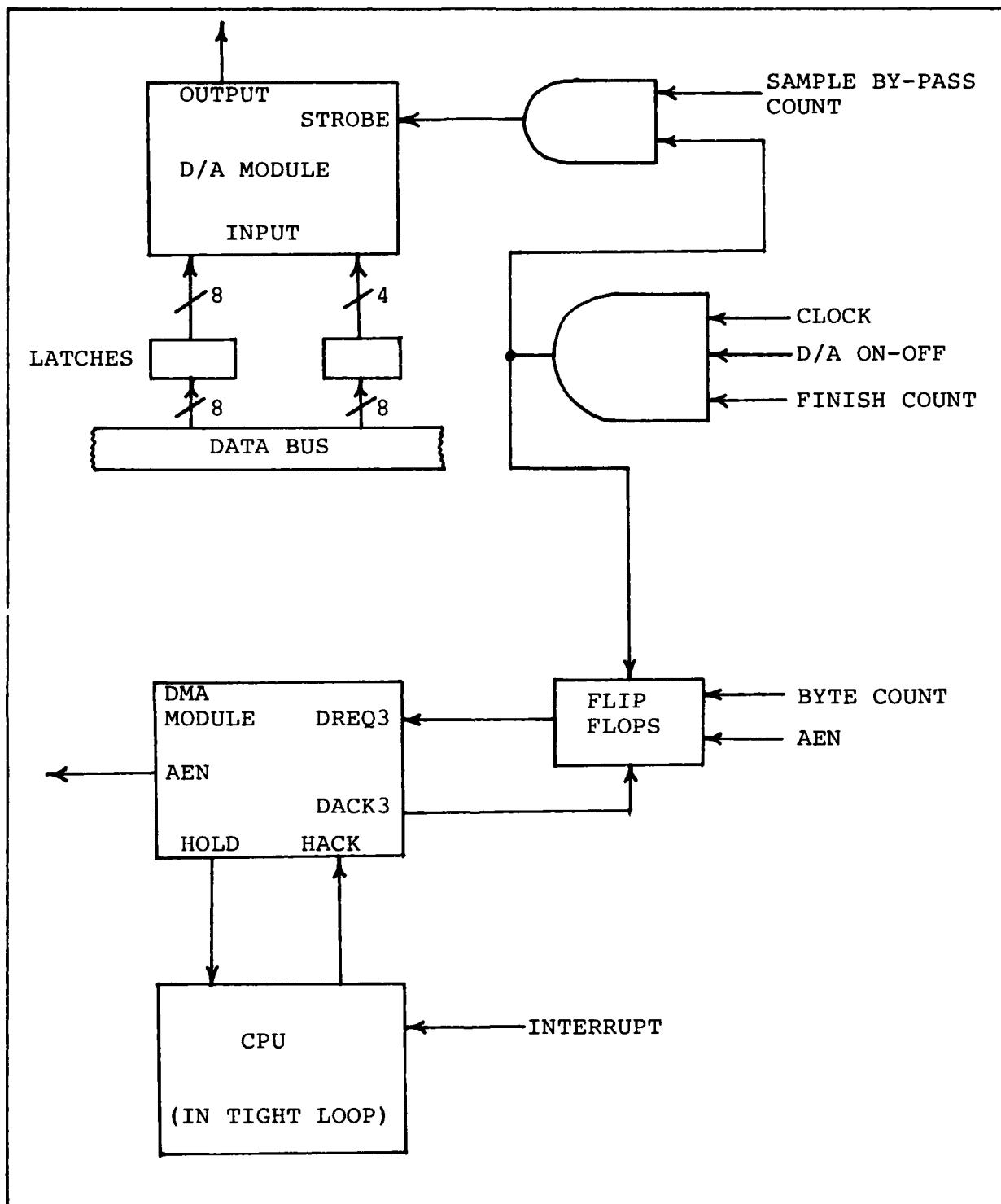


FIG II-2. DIGITAL-TO-ANALOG CONVERSION

this condition until the CPU's interrupt line is made active. This occurs when the user requested number of data samples has been converted. The system user is then returned to the main SPEECH menu.

The MB64 Static RAM Memory Board. The SSM MB64K static Random Access Memory (RAM) by Microcomputer Products, is used to provide an extended addressing memory capability. This extended memory capability is used to store the sampled analog input signal. A maximum of 163,840 samples (327,680 bytes -- two bytes per sample) may be stored. See Figure II-3 for the association of which data samples are stored on which of five extended memory boards. Figure II-4 shows the scheme for the extended memory addressing. Address bits A0 to A15 are controlled by the DMA controller and an external latch. The extended address bits A16 to A19 are controlled by an external counter. The counter is initialized by making LOAD active and placing the initial address on the appropriate data bus lines. The DMA controller end-of-process (EOP) pin then increases the count after each complete pass of addressess A0 to A15. One problem did surface because the 64K static memory boards were used. The problem arose from the original Cromemco 64K memory board and the original boot-up technique associated with it. A graphical representation of the problem is shown in Figure II-5. The original Cromemco 64K memory board has

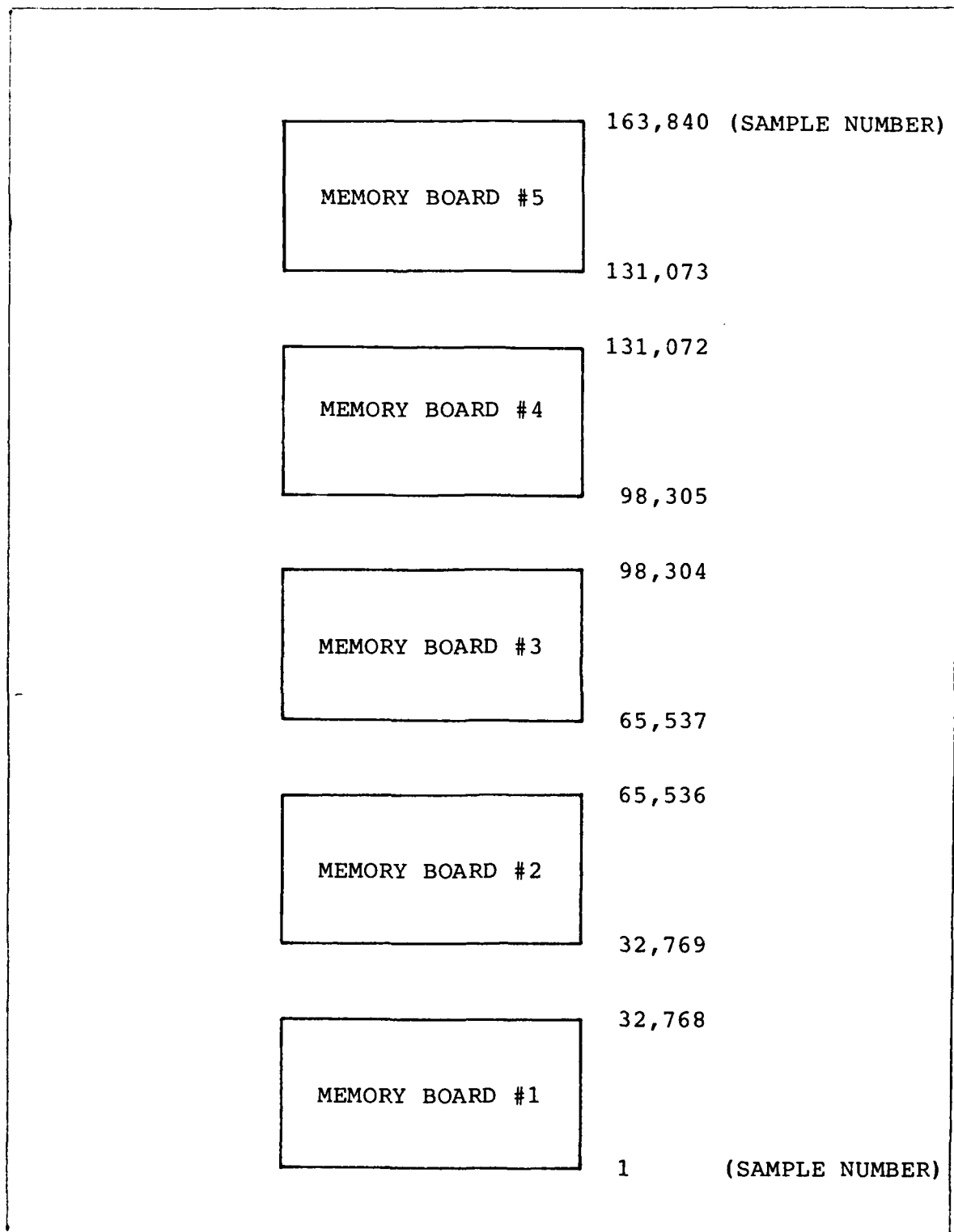


FIG II-3. DATA SAMPLES STORAGE LOCATIONS

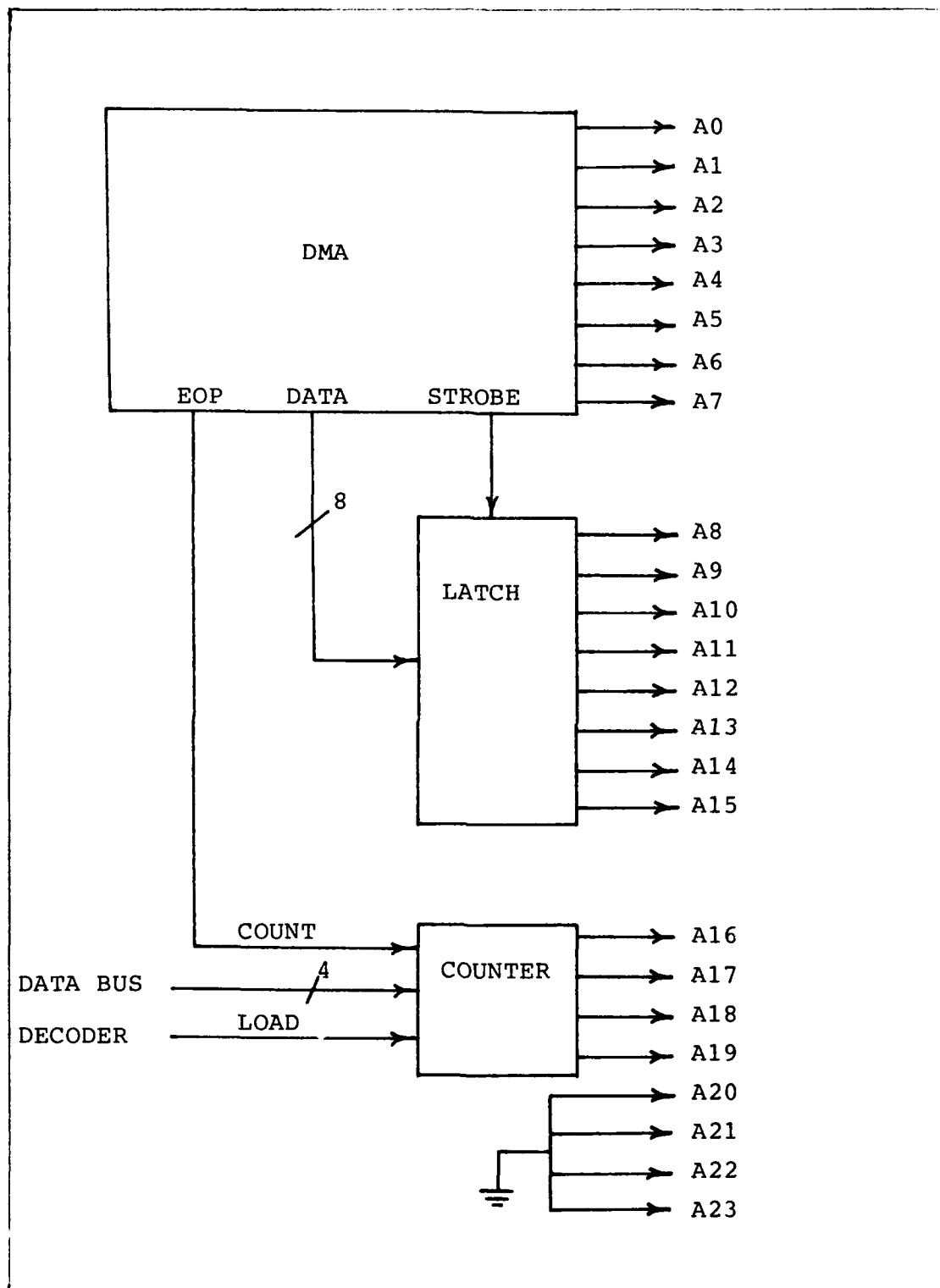


FIG II-4. EXTENDED MEMORY ADDRESSING

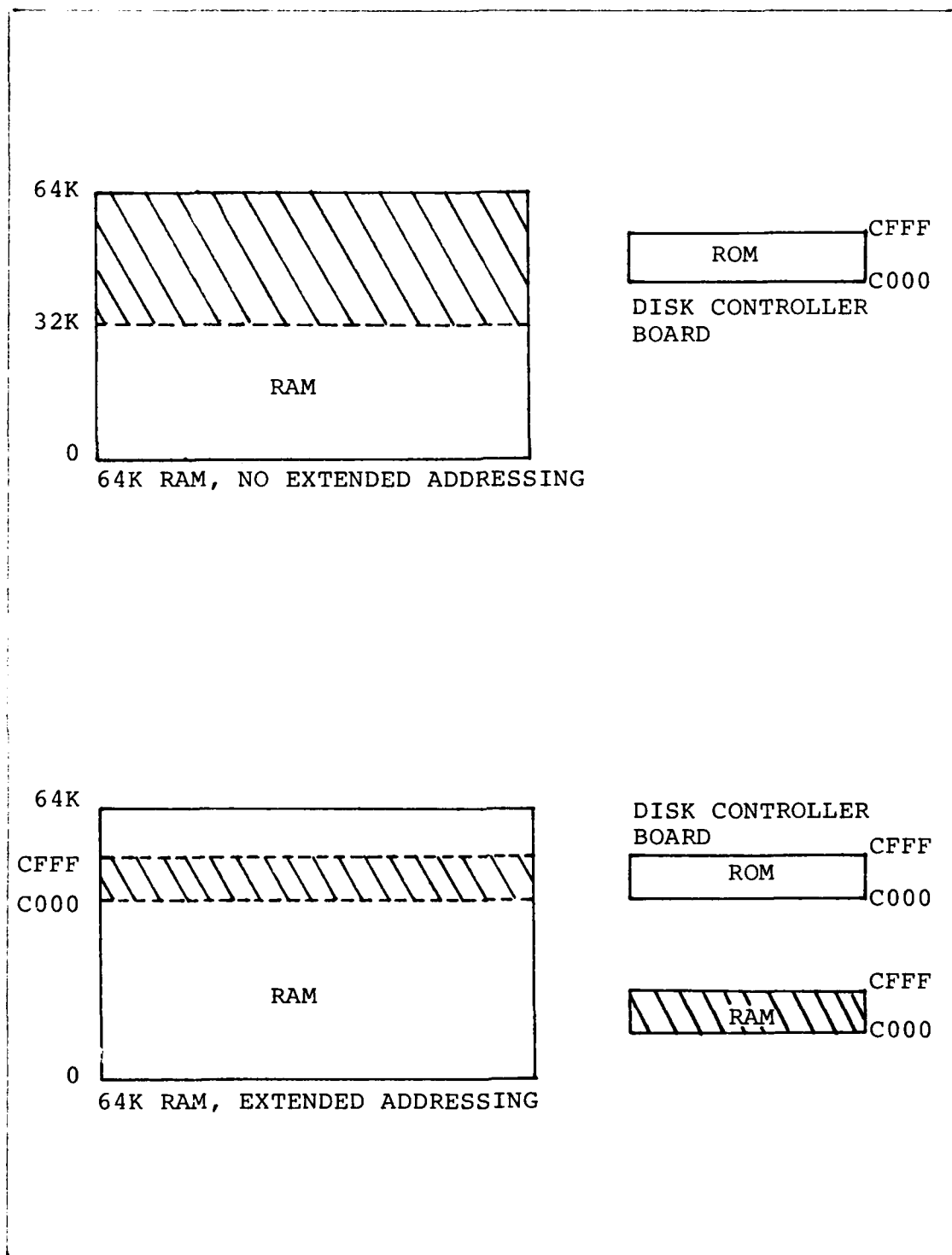


FIG II-5. CROMEMCO BOOT-UP

no capability for extended addressing. This meant replacing the original memory board with one of the new 64K boards. The problem is the original board contains the capability to disable all memory addressed above 32K during system boot-up. This allows the read-only-memory (ROM) located on the disk controller board, at the memory addresses between C000 and CFFF, to be used to boot-up the system. Part of the boot-up program, located on the outer tracks of the system disk, then turns off the ROM memory and turns on the RAM memory above 32K. The new memory boards has one feature which is used to allow system boot-up. By pulling any 2K memory chip off a new board, those memory locations are disabled and ignored. By selecting and removing the two 2K memory chips which cover the memory addresses C000 thru CFFF, the original boot-up technique worked. The technique used is to mount the two removed 2K memory chips on a separate board. The memory chips are set to be inactive at boot-up. The circuitry on the board then detects the command to turn the ROM off. When this command is generated, both removed 2K memory chips are enabled. Thus, all 64K of the system memory is available for use, however, two of the memory chips are now located on a separate board. All of this is transparent to the system user. More specific details on the MB64 Static RAM memory board are contained in Appendix J.

## Software Development

The programming language "C" is used to implement this thesis project. The reasons why are: (1) "C" is a portable programming language. A "C" program written on one computer can be run with little or no modification on any other computer with a "C" compiler. (2) "C"s speed of execution is fast. The difference in execution speed between "C" and an assembler is almost unnoticable. (3) "C" is well suited to structured programming techniques. The formatted structure of "C" allows the software program to be broken down into smaller increments which can be easily handled. There are several sources of information on the "C" language (Ref 8 and 9).

Two problems did arise during the software development. First, it became necessary to incorporate small portions of machine language code into the "C" language program. Additional information on doing this is contained in Appendix L. The inclusion of machine code is required to access the interrupt capability of the Cromemco computer. The basic concept is to start the sampling or conversion process and let it run free under control of the DMA controller. While the DMA controller is in charge, the central processing unit of the Cromemco is placed in a tight loop doing nothing but waiting for an interrupt to occur. The interrupt signal is generated by the STC chip as it reaches the number of last sample to be processed. The specifics of adding machine code to a "C" language program are as follows:

NOTE: The following procedure uses the following "C"  
compiler: BD Software "C" Compiler V1.46  
Copyright (c) 1981 by Leor Zolman

1. The person doing the programming starts with CASM.C, CLINK.COM, CC1.COM and CC2.COM files on their disk. The programmer then enters: CC1 CASM.C <CR>
2. A CASM.CRL file is generated.
3. The programmer then enters: CLINK CASM <CR>
4. A CASM.COM file is generated.
5. The programmer then creates a machine code program by using the CASM guidelines. The machine code program is labeled "FILENAME.CSM", say "FUNCTION.CSM" for this example.
6. The programmer then enters: CASM FUNCTION <CR>
7. A FUNCTION.ASM file is generated.
8. The programmer then enter: MAC FUNCTION \$\$SZPZ.
9. This generates a FUNCTION.HEX file and provides the programmer with the number of sectors to be saved later on the programmers' disk.
10. The programmer then enters: DEBUG FUNCTION.HEX  
(This loads the file "FUNCTION.HEX" into memory.)
11. The programmer then enters: ^C
12. The programmer then enters: SAVE FUNCTION.CRL N  
(Where N is the number of sectors to be saved.)
13. Finally, the programmer enters:  
CLINK "FILENAME OF "C" PROGRAM" FUNCTION

The second problem arose from the need to manipulate numbers which are larger than 16 binary digits. The solution is to use the 32-bit integer package for "C" language programs. More specific details on this mathematics package is contained in Appendix M.

Although the specific software programs are available in Appendix A, it is of value to know which addresses are used to control chip select functions. The software port addresses and their functions are shown in Figure II-6.

Invisible to the system user, the DMA and STC chips are software programmed during the use of the SPEECH program. Figure II-7 shows how the DMA COMMAND REGISTER is loaded during analog-to-digital or digital-to-analog operations. Figure II-8 shows how the DMA COMMAND REGISTER is loaded during memory-to-memory transfer operations. Figure II-9 shows how the DMA COMMAND REGISTER is loaded during the clearing of memory operations. Figure II-10 shows how the DMA CHANNEL 0 MODE REGISTER is loaded during memory-to-memory transfer operations. Channel 0 is used to control the memory SOURCE location during memory-to-memory transfer operations. Figure II-11 shows how the DMA CHANNEL 1 MODE REGISTER is loaded during memory-to-memory transfer operations. Channel 1 is used to control the memory DESTINATION location during memory-to-memory transfer operations. Figure II-12 shows how the DMA CHANNEL 2 MODE REGISTER is loaded during analog-to-digital conversion operations. Channel 2 is used to control the placing of digitized information into static memory for storage.

*****			
PIN	ADDRESS	FUNCTION	
NUMBER	(HEX)		
*****			
1	00	Reserved for use by 16FDC Disk Controller.	
2	80	Reserved for use by TU-ART Device B.	
3	40	Reserved for use by Memory Bank Select. Also turns ROM off.	
4	C0	CLEAR Thesis hardware command.	
5	20	Reserved for use by TU-ART Device A.	
6	A0	Used to activate A-to-D MUX Channel select latch.	
7	60	Used to toggle A-to-D ON/OFF switch.	
8	E0	Reserved for use by processor. Status Port = ED. Data Port = E8.	
9	10	Used as Chip Select for STC chip.	
10	90	Used as Chip Select for DMA chip.	
11	50	Reserved for use by CROMEMCO printer interface.	
13	D0	Used to toggle between CPU and MEM-to-MEM transfer operations.	
14	30	Reserved for use by 16FDC Disk Controller.	
15	B0	Used to toggle between HI-to-LO or LO-to-HI MEM-to-MEM transfers.	
16	70	Used to Chip Select the HI address counter.	
17	F0	Used to toggle D-to-A ON/OFF switch.	
*****			

FIG II-6. Decoder Chip's Port Addresses and Functions.

Figure II-13 shows how the DMA CHANNEL 3 MODE REGISTER is loaded during digital-to-analog conversion operations. Channel 3 is used to control the retrieving of digitized data FROM static memory storage. Figure II-14 shows how the STC MASTER MODE REGISTER is loaded during program operation. Figure II-15 shows how the STC OUT1 COUNTER MODE REGISTER is loaded during program operation. OUT1 is used in conjunction with OUT2 to determine when the analog-to-digital or digital-to-analog conversion processes should be stopped because the total number of desired samples has been taken. Figure II-16 shows how the STC OUT2 COUNTER MODE REGISTER is loaded during program operation. Figure II-17 shows how the STC OUT4 COUNTER MODE REGISTER is loaded during program operation. OUT4 is used to disable digital-to-analog output until a specified number of samples has been by-passed. This is necessary because the output technique always starts with the sample located at the beginning of a memory board regardless of which sample on the board is requested as the first. Figure II-18 shows how the STC OUT5 COUNTER MODE REGISTER is loaded during program operation. OUT5 produces the clock sampling pulses required to sample at the system user specified sampling rate.

A/D or D/A COMMAND REGISTER = (80)HEX

```
*****
*       *       *       *       *       *       *       *
*  7   *  6   *  5   *  4   *  3   *  2   *  1   *  0   *
*       *       *       *       *       *       *       *
*****
*       *       *       *       *       *       *       *
*  1   *  0   *  0   *  0   *  0   *  0   *  X   *  0   *
*       *       *       *       *       *       *       *
*****
```

Bit 0 = 0 ; Memory to Memory transfer disabled.  
 Bit 1 = X ; Does not matter, if bit 0 = 0.  
 Bit 2 = 0 ; Controller enabled.  
 Bit 3 = 0 ; Normal timing.  
 Bit 4 = 0 ; Fixed Priority.  
 Bit 5 = 0 ; Late write selection.  
 Bit 6 = 0 ; DREQ sense set to Active HIGH.  
 Bit 7 = 1 ; DACK sense set to Active HIGH.

FIG II-7. A-to-D or D-to-A COMMAND REGISTER

MEM-to-MEM COMMAND REGISTER = (81)HEX

```
*****
*       *       *       *       *       *       *       *
*  7   *  6   *  5   *  4   *  3   *  2   *  1   *  0   *
*       *       *       *       *       *       *       *
*****
*       *       *       *       *       *       *       *
*  1   *  0   *  0   *  0   *  X   *  0   *  0   *  1   *
*       *       *       *       *       *       *       *
*****
```

Bit 0 = 1 ; Memory to Memory transfer enabled.  
 Bit 1 = 0 ; Channel 0 address hold disables.  
 Bit 2 = 0 ; Controller enabled.  
 Bit 3 = X ; Does not matter, if bit 0 = 1.  
 Bit 4 = 0 ; Fixed Priority.  
 Bit 5 = 0 ; Late write selection.  
 Bit 6 = 0 ; DREQ sense set to Active HIGH.  
 Bit 7 = 1 ; DACK sense set to Active HIGH.

FIG II-8. MEM-to-MEM COMMAND REGISTER

CLEAR MEMORY COMMAND REGISTER = (83)HEX

```

*****
*       *       *       *       *       *       *       *
*  7   *  6   *  5   *  4   *  3   *  2   *  1   *  0   *
*       *       *       *       *       *       *       *
*****
*       *       *       *       *       *       *       *
*  1   *  0   *  0   *  0   *  X   *  0   *  1   *  1   *
*       *       *       *       *       *       *       *
*****

```

Bit 0 = 1 ; Memory to Memory transfer enabled.  
 Bit 1 = 1 ; Channel 0 address hold enabled.  
 Bit 2 = 0 ; Controller enabled.  
 Bit 3 = X ; Does not matter, if bit 0 = 1.  
 Bit 4 = 0 ; Fixed Priority.  
 Bit 5 = 0 ; Late write selection.  
 Bit 6 = 0 ; DREQ sense set to Active HIGH.  
 Bit 7 = 1 ; DACK sense set to Active HIGH.

FIG II-9. CLEAR MEMORY COMMAND REGISTER

CHANNEL 0 MODE REGISTER = (88)HEX

For MEM-MEM transfer---the SOURCE.

```
*****
*       *       *       *       *       *       *
*  7   *  6   *  5   *  4   *  3   *  2   *  1   *  0   *
*       *       *       *       *       *       *
*****
*       *       *       *       *       *       *
*  1   *  0   *  0   *  0   *  1   *  0   *  0   *  0   *
*       *       *       *       *       *       *
*****
```

Bits 1&0 = 00 ; Channel 0.

Bits 3&2 = 10 ; Read transfer.

Bit 4 = 0 ; Autoinitialize disabled.

Bit 5 = 0 ; Address increment selected.

Bits 7&6 = 10 ; Block mode selected.

FIG II-10. CHANNEL 0 MODE REGISTER

CHANNEL 1 MODE REGISTER = (85)HEX

For MEM-MEM transfer---the DESTINATION.

```
*****
*       *       *       *       *       *       *
*  7   *  6   *  5   *  4   *  3   *  2   *  1   *  0   *
*       *       *       *       *       *       *
*****
*       *       *       *       *       *       *
*  1   *  0   *  0   *  0   *  0   *  1   *  0   *  1   *
*       *       *       *       *       *       *
*****
```

Bits 1&0 = 01 ; Channel 1.

Bits 3&2 = 01 ; Write transfer.

Bit 4 = 0 ; Autoinitialize disabled.

Bit 5 = 0 ; Address increment selected.

Bits 7&6 = 10 ; Block mode selected.

FIG II-11. CHANNEL 1 MODE REGISTER

CHANNEL 2 MODE REGISTER = (56)HEX  
For Analog-to-Digital Conversions

```
*****
*       *       *       *       *       *       *       *
*  7   * 6   * 5   * 4   * 3   * 2   * 1   * 0   *
*       *       *       *       *       *       *       *
*****
*       *       *       *       *       *       *       *
*  0   * 1   * 0   * 1   * 0   * 1   * 1   * 0   *
*       *       *       *       *       *       *       *
*****
```

Bits 1&0 = 10 ; Channel 2.  
Bits 3&2 = 01 ; Write transfer.  
Bit 4 = 1 ; Autoinitialize enabled.  
Bit 5 = 0 ; Address increment selected.  
Bits 7&6 = 01 ; Single mode selected.

FIG II-12. CHANNEL 2 MODE REGISTER

CHANNEL 3 MODE REGISTER = (5B)HEX  
For Digital-to-Analog Conversions

```
*****
*       *       *       *       *       *       *       *
*  7   * 6   * 5   * 4   * 3   * 2   * 1   * 0   *
*       *       *       *       *       *       *       *
*****
*       *       *       *       *       *       *       *
*  0   * 1   * 0   * 1   * 1   * 0   * 1   * 1   *
*       *       *       *       *       *       *       *
*****
```

Bits 1&0 = 11 ; Channel 3.  
Bits 3&2 = 10 ; Read transfer.  
Bit 4 = 1 ; Autoinitialize enabled.  
Bit 5 = 0 ; Address increment selected.  
Bits 7&6 = 01 ; Single mode selected.

FIG II-13. CHANNEL 3 MODE REGISTER

MASTER MODE REGISTER = (C10E)HEX  
 Command Code Address = (17)HEX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

1	1	0	0	0	0	0	1	0	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Bits 3-0 = 1110 ; Set for 32 bit compare.  
 Bits 7-4 = 0000 ; Set FOUT SOURCE = F1.  
 Bits 11-8 = 0001 ; Set FOUT Divided by 1.  
 Bit 12 = 0 ; Set FOUT on.  
 Bit 13 = 0 ; Set to support an 8 bit bus.  
 Bit 14 = 1 ; Set Disable Increment.  
 Bit 15 = 1 ; Set Scalar Control to BCD Division.

FIG II-14. MASTER MODE REGISTER

OUT 1 COUNTER MODE REGISTER = (0029)HEX  
 Command Code Address = (01)HEX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Bits 2-0 = 001 ; Active HIGH pulse out.  
 Bit 3 = 1 ; Count up.  
 Bit 4 = 0 ; Binary count.  
 Bit 5 = 1 ; Count repetitively.  
 Bit 6 = 0 ; Reload from LOAD.  
 (Note: Load LOAD with (00)HEX.)  
 Bit 7 = 0 ; Disable special gate.  
 Bits 11-8 = 0000 ; Count Source = (TCN-1).  
 Bit 12 = 0 ; Count on rising edge.  
 Bits 15-13 = 000 ; No gating.

FIG II-15. OUT 1 COUNTER MODE REGISTER

OUT 2 COUNTER MODE REGISTER = (002D)HEX  
 Used in COUNT FINISHED control.  
 Command Code Address = (02)HEX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	1

Bits 2-0 = 101 ; Active LOW pulse out.  
 Bit 3 = 1 ; Count up.  
 Bit 4 = 0 ; Binary count.  
 Bit 5 = 1 ; Count repetitively.  
 Bit 6 = 0 ; Reload from LOAD.  
 (Note: Load LOAD with (00)HEX.)  
 Bit 7 = 0 ; Disable special gate.  
 Bits 11-8 = 0000 ; Count Source = (TCN-1).  
 Bit 12 = 0 ; Count on rising edge.  
 Bits 15-13 = 000 ; No gating.

FIG II-16. OUT 2 COUNTER MODE REGISTER

OUT 4 COUNTER MODE REGISTER = (1402)HEX  
 Used for Digital-to-Analog OFFSET control  
 Command Code Address = (04)HEX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0

Bits 2-0 = 010 ; Active HIGH toggle (delayed).  
 Bit 3 = 0 ; Count down.  
 Bit 4 = 0 ; Binary count.  
 Bit 5 = 0 ; Count once.  
 Bit 6 = 0 ; Reload from LOAD.  
 Bit 7 = 0 ; Disable special gate.  
 Bits 11-8 = 0100 ; Count Source = SOURCE 4.  
 Bit 12 = 1 ; Count on falling edge.  
 Bits 15-13 = 000 ; No gating.

FIG II-17. OUT 4 COUNTER MODE REGISTER

OUT 5 COUNTER MODE REGISTER = (0B21)HEX  
 Used for CLOCK PULSES.  
 Command Code Address = (05)HEX

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1

Bits 2-0 = 001 ; Active HIGH pulse out.  
 Bit 3 = 0 ; Count down.  
 Bit 4 = 0 ; Binary count.  
 Bit 5 = 1 ; Count repetitively.  
 Bit 6 = 0 ; Reload from LOAD.  
 Bit 7 = 0 ; Disable special gate.  
 Bits 11-8 = 1011 ; Count Source = Fl.  
 Bit 12 = 0 ; Count on rising edge.  
 Bits 15-13 = 000 ; No gating.

FIG II-18. OUT 5 COUNTER MODE REGISTER

## Graphics

The graphical display of digitized data samples is made possible by using the Imaginator, built by the Cleveland Codonics Incorporated. The Imaginator is an intelligent, high resolution graphics retrofit unit. The Imaginator has its own onboard microcomputer to perform graphics processing independent of the host computer. The graphical display is placed within the 504 pixel by 247 pixel resolution of the Imaginator. More specific information on the Imaginator is contained in Appendix K. Figure II-19 shows the basic grid display. As noted on the figure, 500 samples at a time are displayed. This is accomplished by locating the 1000 bytes which define the 500 samples requested by the system user. The 1000 bytes are brought into a buffer array in lower memory (under 64K). Here the data is manipulated into a range from plus to minus five volts. The resulting data is then plotted on the grid. The ability to move both the vertical and horizontal cursors is provided. A START sample number and a FINISH sample number are selectable by the system user. These data samples may then be output to an external analog system, for example, a speaker.

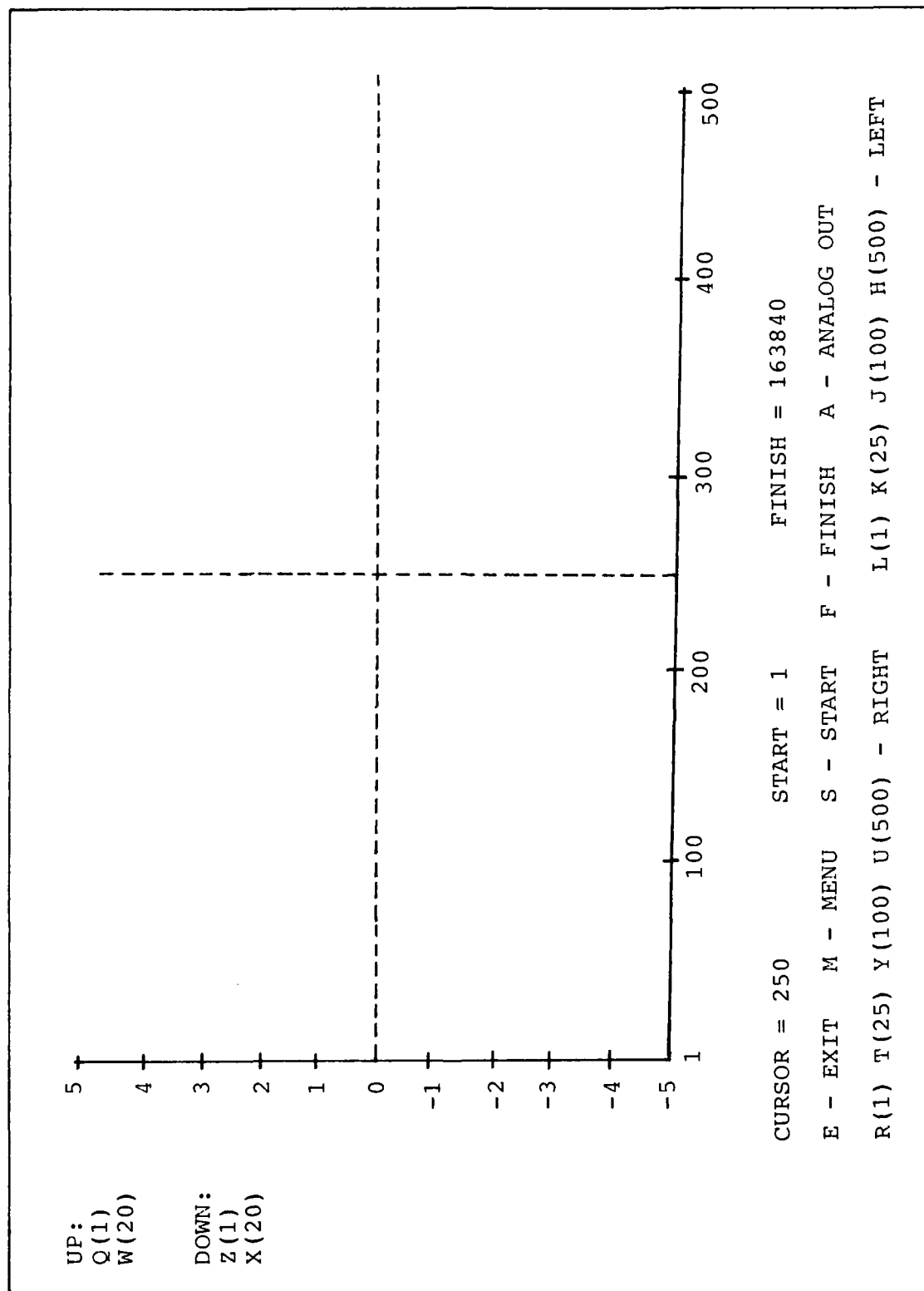


FIG II-19. GRAPHICAL DISPLAY FORMAT

### III. Design and Fabrication

This chapter briefly describes the design and fabrication of both the hardware and software associated with the this thesis project.

#### Hardware

The hardware design resulted in the development of four separate S-100 boards. The four boards are the "Boot Up Control", the "Direct Memory Access", the "Digital-to-Analog Conversion", and the "Analog to Digital Conversion" boards. Two types of standard S-100 boards were used. When possible a S-100 board with power and ground busses already installed is used. These boards are used when only standard sized integrated circuit chips -- mounted in wire wrap sockets -- plus standard voltage regulators, are mounted on the board. When larger devices required mounting, such as the analog-to-digital conversion device, then a plain board is used. In this case, all voltage and ground paths are also added to the board. As implied above, a wire wrap/then soldering technique is used for all board construction. It is necessary at times to connect specific signals between two fabricated boards. Where possible, the S-100 bus positions are used. In the cases where S-100 bus positions are not available, a ribbon cable is made and used between the two boards in question to allow data or sometimes power access.

In addition to the fabricated boards, the commercially available MB64K static memory boards, discussed in chapter two, are also installed onto the S-100 bus motherboard.

The integrated circuit layout on the boards is clearly shown in Appendix B. The "Boot Up Control" board integrated circuit layout is shown on Appendix B, page B-1. The "Direct Memory Access" board integrated circuit layout is shown on Appendix B, page B-2. The "Digital-to-Analog Conversion" board integrated circuit layout is shown on Appendix B, page B-3. And finally, the "Analog-to-Digital Conversion" board integrated circuit layout is shown on Appendix B, page B-4.

The cross reference between the integrated circuit number, shown on the integrated circuit layouts, and what integrated circuit is really used, is shown in Appendix C, starting on page C-1.

The major portion of the hardware fabrication is shown in the circuit wiring diagrams of Appendix D. The "Boot Up Control" wiring diagram is shown on Appendix D, page D-1. The "Direct Memory Access" wiring diagram is shown on Appendix D, pages D-2 thru D-4. The "Digital-to-Analog" wiring diagram is shown on Appendix D, page D-5. And finally, the "Analog-to-Digital" wiring diagram is shown on Appendix D, page D-6.

Appendix E provides the timing diagrams used during the thesis project. Appendix E, page E-1 displays the timing diagram for memory-to-memory transfers. Appendix E, page E-2 displays the timing diagram for digital-to-analog

conversions. And finally, Appendix E, page E-3 displays the timing diagram for analog-to-digital conversions.

### Software

Due to the size and complexity of the software programs required by this thesis project, a structured programming format is used. In this format, each operation is broken down into simple programming segments which are executed sequentially or separately as needed to implement a desired operation. What follows is a description of each program segment and its development.

SPEECH.C. This is the entry module for reaching all other program segments. Extremely short in size but also extremely important in its function, SPEECH.C causes several things to occur. First, the SPEECH title is displayed on the video display screen. Thus, the system user knows the SPEECH program has been correctly accessed. Second, an introduction is provided on the video display screen for the system user. This is part of making the program user friendly. Third, this program segment sets the default values for all global variables used in any of the program segments. Finally, this program segment provides a menu of "Command Options" to be displayed to the system user. (Appendix A, page A-1)

SPEECH.H. This program segment is a "C" language header. It contains the GLOBAL VARIABLES used in the entire SPEECH program. SPEECH.H is included as part of each individual "C" module. Any change to SPEECH.H requires all "C" modules to be recompiled, via the CCl command, and relinked, via the CLINK command, to form a new SPEECH.COM file. (Appendix A, page A-3)

TITLE.C. When called, this program displays the program title -- SPEECH -- and the program author's name on the video display screen. (Appendix A, page A-6)

INTRO.C. An introduction to the system user is displayed when this program segment is activated. This segment asks for and stores the name of the system user for future display by other program segments. The system user is also asked if the entire SPEECH program description is to be displayed on the video display screen, and responds accordingly to a yes or no response. (Appendix A, page A-8)

DESCRIBE.C. This program segment displays a description of the SPEECH program and a user's guide on the video display screen. (Appendix A, page A-10)

DEFAULTS.C. The activation of this program segment automatically initializes specified GLOBAL VARIABLES to default values for use by other program segments. (Appendix A, page A-17)

MENU.C. This program segment prompts the system user with a list of available "Command Options". The option chosen by the system user is then implemented. (Appendix A, page A-20)

QUIT.C. A goodbye message to the system user is displayed on the video display screen when this program segment is addressed. The SPEECH program is then exited and the system user is returned to the operating system. (Appendix A, page A-23)

ANALOG.C. This program segment controls the analog-to-digital conversion of an analog input signal such as speech. The system user selects the "analog channel" to be sampled (1 of 16), the data "sampling rate", and the total "number of samples" to be taken. Options in this segment allow the system user to exit the program, return to the system MENU, or activate a bell which signals the beginning of the sampling process. (Appendix A, page A-24)

DIGITAL.C. This program segment controls the digital-to-analog conversion of data samples stored in the extended memory of the Cromemco S-100 microcomputer system. The system user selects the first and last sample to be converted. Options in this segment allow the system user to exit the program, return to the system MENU, or activate a bell which signals the start of the conversion process. (Appendix A, page A-27)

STORE.C.        The transfer of a system user specified number of data samples is controlled by this program segment. The transfer is from the extended static random access memory (RAM) to magnetic storage disk. The data transfer is to a system user specified "filename". The default "filename" is: B:DATA.ONE. Data transfer is done 2048 bytes at a time by using the array named mem\_buffer. (Appendix A, page A-30)

RETRIEVE.C       This program segment clears the extended static RAM and then transfers data from magnetic disk storage to extended memory. The data transfer is from a system user specified "filename". The default "filename" is: B:DATA.ONE. Data transfer is done 2048 bytes at a time by using the array named mem\_buffer. Data transfer is continued as long as data remains on the magnetic disk under the "filename" specified. (Appendix A, page A-34)

GRAPHICS.C.       This program segment controls the graphical display of 500 data samples at a time on the video display screen. The segment displays user prompts, provides right and left cursor movement, and provides up and down volt-line movement. While observing the graphical display, the system user may the vertical cursor to select the starting and finishing data sample numbers for use during a later digital-to-analog output. (Appendix A, page A-38)

PLOT.C.        This program segment draws the horizontal axis, the vertical axis, and the zero volt line of the

graphical display. It also transfers the 500 data samples (1000 bytes) to be displayed. The program segment then calculates the y-axis values of the 500 data samples and then plots the 500 data samples on the graph. Finally, the system user is returned to the calling program segment. (Appendix A, page A-44)

RIGHT.C. This program segment moves the vertical cursor displayed in program segment PLOT.C to the right by an amount specified by the system user. Any request for movement beyond sample number 500 causes a new 500 sample graph to be displayed. The cursor is then placed at the first data sample of the new data plot. The vertical cursor is not allowed to move beyond the last sample of all samples taken. (Appendix A, page A-48)

LEFT.C. This program segment moves the vertical cursor displayed in program segment PLOT.C to the left by an amount specified by the system user. Any request for movement beyond sample number one of 500 samples causes a new 500 sample graph to be displayed. The cursor is then placed at data sample 500 of the new data plot. The vertical cursor is not allowed to move beyond sample number 1 of all samples taken. (Appendix A, page A-51)

VOLTLINE.C. This program segment move the horizontal volt-line cursor up or down by an amount specified by the system user. The volt-line is not allowed to move off of the graph. (Appendix A, page A-54)

INPUT.C. This program segment is used to allow the system user to: (1) select the "analog input channel" to be sampled, (2) select the data "sampling rate", (3) select the total "number of samples" to be taken, and (4) select the "first and last sample" to be converted during digital-to-analog conversions. (Appendix A, page A-55)

DMA.C. This program segment initializes the direct memory access (DMA) chip's internal registers to: (1) support analog-to-digital sampling, (2) support digital-to-analog conversion, (3) support 2048 byte memory-to-memory data transfer in support of STORE.C and RETRIEVE.C, and (4) support 64K byte memory-to-memory data transfer to clear the extended static memory RAM boards. (Appendix A, page A-61)

TIMING.C. This program segment initializes the system timing controller (STC) chip's internal registers to: (1) provide analog-to-digital or digital-to-analog clock pulses at a system user specified triggering rate, (2) disable analog-to-digital or digital-to-analog capability after a system user specified number of data samples have been processed, and (3) provide by-passing of unrequired data samples at the beginning of a memory board during digital-to-analog output operations. (Appendix A, page A-65)

CLEARMEM.C. Binary zero (00000000B) is placed in each memory byte of all extended memory boards when this program segment is addressed. (Appendix A, page A-70)

NOP.CSM. One of two non "C" language programs used in the thesis project, this program segment provides the delay required during loading of the DMA chip's internal registers. (Appendix A, page A-73)

WAIT.CSM. One of two non "C" language programs used in the thesis project, this program segment prevents the central processing unit from executing SPEECH program statements until: (1) all analog-to-digital data samples have been taken, or (2) all digital-to-analog data samples have been output. This prevention is accomplished by placing the central processing unit in a tight loop. The loop is not exited until the interrupt signal occurs. The system timing controller chip activates the interrupt signal after all samples have been taken or have been used, as the case may be. (Appendix A, page A-74)

#### IV. Validation

The validation of this project is limited to four points.

First, the method developed to boot-up the system is demonstrated to work. The boot-up control board works perfectly and required only a few initial bugs to be worked out of the board.

Second, much of the software is demonstrated to work. The graphics programs works exceptionally well. The display grid is easily displayed. The grid numbering system is automatically updated as requests for data both above and below the currently displayed data are made. The cursor control commands are successfully demonstrated both for small (1 pixel), medium, and large (500 pixel) cursor movements. Data is easily transferred both to magnetic disk storage and back again. The disappointing part is that canned data had to be displayed on the grid and transferred to magnetic disk because some of the circuitry did not work. Specifically, the control of data sampling, controlled by the system timing controller chip, did not function. The registers within the chip which controlled the data addressing often clocked at the wrong rate. Also the registers which signalled the completion of a sampling session failed to function.

Third, the decoder circuit used to set conditions is demonstrated to work. This is done by using an oscilloscope

to display logic high or logic low levels on designated integrated circuit chip pins. The oscilloscope also displayed that logic levels are correct during requests for analog-to-digital conversions and for digital-to-analog conversions. The weak link is related to the malfunction of the registers within the system timing controller chip.

Fourth, the extended memory addressing is function properly. This is seen by observing the extended memory address LED's on the S-100 bus plug-in monitor board when a sampling command is given.

## V. Conclusions and Recommendations

### Conclusions

The basic goal of sampling speech signals and displaying the sampled signals on the video display screen is not met. The system timing chip appeared to be the weak link. Without changing any connections or making any software changes, the system timing chip changed sampling frequency. It is also believed that the register functions within the system timing chip are not operating correctly. Due to a lack of a second timing chip and with only a few days left in the quarter, further investigation into the locking up of the data sampling program is not included in this document.

On the positive side, much of the designed hardware and software is known to work very well. The modification to allow system boot-up with extended memory is working perfectly. The extended memory addressing is shown to work by observing the address LED's on the S-100 plug-in monitor board. The graphics display programs run well, even though the data they use is "canned" data. Data is easily transferred to and from magnetic disk storage. Command locations for sending data to the NOVA computer system are left available in the software program for future use.

The "C" programming language with its modular format is demonstrated to be very powerful, friendly and easy to use.

### Recommendations

This project still is full of merit and should be persued and completed. It is recommended, however, that all persons who do follow-on work on this thesis project have previous hardware experience. The lack of hardware experience is a major cause for delays in the fabrication and understanding of the circuit boards. Individuals with previous hardware experience are invited to investigate the manner in which the extended memory addressing is accomplished. Some chip reduction in this area is likely to be possible.

## Bibliography

1. Sargent, Murray III and Shoemaker, Richard L.  
Interfacing Microcomputers to the Real World. Reading  
Massachusetts: Addison-Wesley Publishing Company, 1981.
2. 023-0012. Cromemco ZPU Z-80 Central Processing Unit.  
Instruction Manual. Mountain View California: Cromemco  
Incorporated, 1978.
3. Cromemco Z80A Central Processing Unit. Technical  
Manual. Mountain View California: Cromemco  
Incorporated, 1980.
4. 023-2004. Cromemco 16FDC Floppy Disk Controller.  
Instruction Manual. Mountain View California: Cromemco  
Incorporated, 1980.
5. 023-0008. 64K Random Access Memory. Instruction  
Manual. Mountain View California: Cromemco  
Incorporated, 1979.
6. 023-0036. Disk Operating System, Series-2 CDOS.  
Instruction Manual. Mountain View California: Cromemco  
Incorporated, 1980.
7. 023-0011. Cromemco TU-ART Digital Interface.  
Instruction Manual. Mountain View California: Cromemco  
Incorporated, 1978.
8. BD Software C Compiler v1.4. User's Guide. Brighton  
Massachusetts: BD Software, 1981.
9. Kernighan, Brian W. and Ritchie, Dennis M. The C  
Programming Language. Englewood Cliffs, New Jersey:  
Prentice-Hall Incorporated, 1978.

```

/*****
/*
/*      NAME:      SPEECH.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 1
/*      FUNCTION:
/*          See user's guide in module #3 (describe) for complete
/*          description of SPEECH. SPEECH.C (module #1) is the entry
/*          module. It (1) displays the program title, (2) provides an
/*          introduction to the user, (3) sets default values for global
/*          variables, and (4) provides a menu of "command options" to
/*          the user.
/*      INPUTS: NONE.
/*      OUTPUTS: NONE.
/*      GLOBAL VARIABLES USED: exit.
/*      GLOBAL VARIABLES CHANGED: NONE.
/*      FILES READ: NONE.
/*      FILES WRITTEN: NONE.
/*      MODULES CALLED: title(), intro(), set_defaults(), menu().
/*      CALLING MODULES: NONE.
/*
/*      AUTHOR: CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY: AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*               Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*      SYMBOLIC CONSTANTS
/*
*****/
#include "bdscio.h"          /* Standard 'C' Language header file. */

```

```

/*****
/*
/*      GLOBAL VARIABLES
/*
*****/

```

```

/* NOTE: All GLOBAL VARIABLES must be listed just prior */
/*        to main() whether used in main() or not.      */

```

```

#include "speech.h"          /* Contains all GLOBAL VARIABLES. */

```

```

/*****
/*
/*                                MAIN PROGRAM                                */
/*
/*
/*****
main()
{
    title();                /* Print "SPEECH" on the CRT.                */
    intro();                /* Give user the option to read the */
                           /* description of "SPEECH".         */
    set_defaults();         /* Initialize variables to default */
                           /* values.                          */
    while(exit == FALSE){   /* Show the user the commands which can */
        menu();             /* implemented. Continue to return to */
        continue;          /* the menu of commands until the user */
    }                       /* wishes to EXIT the program.        */
}

```

```

/*****
/*
/*      NAME:      SPEECH.H
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 2
/*      FUNCTION:
/*          SPEECH.H is a "C Language" Header which holds the GLOBAL
/*          VARIABLES used in the entire SPEECH program.  SPEECH.H is
/*          "# included" in each individual "C" module.  Any change to
/*          SPEECH.H will require all "C" modules to be recompiled
/*          (CC1 command) and relinked (CLINK command) to form a new
/*          SPEECH.COM file.
/*      INPUTS:  NONE.
/*      OUTPUTS: NONE.
/*      GLOBAL VARIABLES USED:  NONE.
/*      GLOEAL VARIABLES CHANGED:  NONE.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  NONE.
/*      CALLING MODULES:  NONE.
/*
/*      AUTHOR:  CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:  AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*               Speech Work Station.  Thesis Advisor: Major Larry Kizer.
/*
/*****

```

```

/*****
/*
/*               SPEECH.H
/*
/*****
int exit;          /* Set TRUE by quit() to exit program.
char name[50];     /* Holds user's name for later display.

int input1;        /* Used to control cursor movement.
int input2;        /* Used to control cursor movement.
int cursor;        /* Holds pixel number of cursor. Pixel number
/*               is one more than sample number.
/*               I.E. Sample #1 is in pixel column #2.

char a_temp[7];    /* A temporary holding place for an ASCII string.
char l_temp[4];    /* A temporary holding place for a LONG (32 bit)
/*               INTEGERS.
char l_cursor[4];  /* Holds LONG (32 bit) version of cursor.
char x_axis[4];    /* Holds LONG (32 bit) version of "sample #"
/*               displayed in first column (pixel #2) of
/*               graph.

```

```

int is_analog;          /* Set TRUE for Analog-to-Digital operations. */
int is_digital;         /* Set TRUE for Digital-to-Analog operations. */
int is_mem_mem;         /* Set TRUE for Memory-to-Memory transfers. */
int is_graphics;        /* Set TRUE for Graphics operations. */
int is_clearmem;        /* Set TRUE for NULLING Extended Memory. */

char filename[18];      /* Holds "filename" used to STORE & RETRIEVE */
                        /* data to magnetic disk. */
char channel[7];        /* Holds ASCII version of ANALOG CHANNEL to */
                        /* be sampled. */
char rate[7];           /* Holds ASCII version of SAMPLING RATE. */
char max_rate[7];       /* Holds ASCII version of MAXIMUM SAMPLING RATE. */
char samples[7];        /* Holds ASCII version of total number of */
                        /* SAMPLES to be taken. */
char max_samples[7];    /* Holds ASCII version of maximum number of */
                        /* samples which can be stored. */
char begin_at[7];       /* Holds ASCII version of first sample to be */
                        /* used during digital-to-analog operations. */
unsigned chan_num;       /* Holds BINARY version of ANALOG CHANNEL to */
                        /* be sampled. */
char rate_num[4];       /* Holds LONG (32 bit) version of SAMPLING RATE. */
char start[4];          /* Holds LONG (32 bit) version of first sample */
                        /* to be used during digital-to-analog */
                        /* operations. */
char finish[4];         /* Holds LONG (32 bit) version of last sample */
                        /* to be used during digital-to-analog */
                        /* operations. */

/*****
/* NOTE: limit1A thru limit8A are used during digital-to-analog */
/* operations to set the initial extended memory board to be used. */
*****/

char limit1A[7];        /* Holds ASCII version of first "sample" stored */
                        /* on extended memory board #1. */
char limit2A[7];        /* Holds ASCII version of last "sample" stored */
                        /* on extended memory board #1. */
char limit3A[7];        /* Holds ASCII version of first "sample" stored */
                        /* on extended memory board #2. */
char limit4A[7];        /* Holds ASCII version of last "sample" stored */
                        /* on extended memory board #2. */
char limit5A[7];        /* Holds ASCII version of first "sample" stored */
                        /* on extended memory board #3. */
char limit6A[7];        /* Holds ASCII version of last "sample" stored */
                        /* on extended memory board #3. */
char limit7A[7];        /* Holds ASCII version of first "sample" stored */
                        /* on extended memory board #4. */
char limit8A[7];        /* Holds ASCII version of last "sample" stored */
                        /* on extended memory board #4. */

char crystal[8];        /* Holds ASCII version of crystal frequency */
                        /* used with STC chip. */

```

```

char mem_buffer[2048]; /* Temporary holding place in lowest 64K of */
                        /* memory for data. Used during disk I/O and */
                        /* plotting operations. */

char from[4];          /* Holds LONG (32 bit) version of first byte to */
                        /* be moved during memory-to-memory transfer. */

char to[4];            /* Holds LONG (32 bit) version of where first */
                        /* byte is to be moved to during memory-to- */
                        /* transfer. */

int volt_line;         /* Used to control up/down VOLT_LINE movement. */

```

```

/*****
/*
/*      NAME:      TITLE.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 3
/*      FUNCTION:  Displays program title (SPEECH) and author's name on
/*                  CRT screen.
/*      INPUTS:    NONE.
/*      OUTPUTS:   Programmed "text" displayed on CRT screen.
/*      GLOBAL VARIABLES USED:  NONE.
/*      GLOBAL VARIABLES CHANGED:  NONE.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  NONE.
/*      CALLING MODULES:  main()
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station.  Thesis Advisor: Major Larry Kizer.
/*
/*
/*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
/*****/
#define ESCAPE      27          /* H-19 CRT ASCII "escape" code.
#define CLEARS      69          /* H-19 CRT ASCII clear the screen code.

```

```

/*****
/*
/*                  FUNCTION:  TITLE()
/*
/*****/
title()
{
    putchar(ESCAPE);          /* Clear CRT screen.
    putchar(CLEARS);          /* Display "SPEECH" on CRT screen.

    printf("\n\n\n\n");
    printf("\t");
    puts("XXXXXXX XXXXXXX XXXXXXX XXXXXXX XX XX\n");
    printf("\t");
    puts("XXXXXXX XXXXXXX XXXXXXX XXXXXXX XX XX\n");
    printf("\t");
    puts("XX XX XX XX XX XX XX\n");
    printf("\t");
    puts("XX XX XX XX XX XX XX\n");
    printf("\t");
    puts("XXXXXXX XXXXXX XXXXX XXXXX XX XXXXXX\n");
    printf("\t");

```

```

puts("XXXXXXXX XXXXXXXX XXXXXX XXXXX XX XXXXXXXX\n");
printf("\t");
puts(" XX XX XX XX XX XX XX\n");
printf("\t");
puts(" XX XX XX XX XX XX\n");
printf("\t");
puts("XXXXXXXX XX XXXXXXXX XXXXXXXX XXXXXXXX XX XX\n");
printf("\t");
puts("XXXXXXXX XX XXXXXXXX XXXXXXXX XXXXXXXX XX XX\n");
/* Display authors name on CRT screen. */
printf("\n\n\n\n\n\n\n\n\n\n");
printf("\t\t\t\t\t by: CAPT WILLIAM H. LIEBER");

sleep(40); /* Wait while user enjoys title.*/
putchar(ESCAPE); /* Clear the CRT screen. */
putchar(CLEAR);
}

```

```

/*****
/*
/*      NAME:      INTRO.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 4
/*      FUNCTION:  Stores user's name for future display.  Asks user if
/*                  program description should be displayed.  Responds to user's
/*                  yes or no response.
/*      INPUTS:    User entered commands from H-19 keyboard.
/*      OUTPUTS:   User prompts displayed on CRT screen.
/*      GLOBAL VARIABLES USED:  name.
/*      GLOBAL VARIABLES CHANGED:  name.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  describe().
/*      CALLING MODULES:  main()
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station.  Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
*****/
#define ESCAPE      27    /* H-19 CRT ASCII "escape" code.
#define CLEARS      69    /* H-19 CRT ASCII clear the screen code.
#define R_VIDEO     0x70  /* H-19 CRT code for enter reverse video mode.
#define N_VIDEO     0x71  /* H-19 CRT code for enter normal video mode.

```

```

/*****
/*
/*                  GLOBAL VARIABLES
/*
*****/
#include "speech.h"          /* Contains all GLOBAL VARIABLES.

```

```

/*****
/*
/*                  FUNCTION:  INTRO()
/*
*****/
intro()
{
    /*
    /*                  LOCAL VARIABLES
    /*
    *****/
    int see_intro;          /* Holds keyboard response to getchar().
    int i;                  /* Indexing variable used in "for" loop.

```



```

/*****
/*
/*      NAME:      DESCRIBE.C
/*      VERSION:   1.0
/*      DATE:      2 Decemober 1983
/*      MODULE NUMBER: 5
/*      FUNCTION:  Displays SPEECH'S description and user's guide on
/*                  the CRT screen.
/*      INPUTS:    User entered commands from H-19 keyboard.
/*      OUTPUTS:   Programmed "text" displayed on CRT screen.
/*      GLOBAL VARIABLES USED:  name.
/*      GLOBAL VARIABLES CHANGED:  NONE.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  NONE.
/*      CALLING MODULES:  main(), menu().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
*****/
#define ESCAPE      27  /* H-19 CRT ASCII "escape" code.
#define CLEARS      69  /* H-19 CRT ASCII clear the screen code.
#define R_VIDEO     0x70 /* H-19 CRT code for enter reverse video mode.
#define N_VIDEO     0x71 /* H-19 CRT code for enter normal video mode.

```

```

/*****
/*
/*                  FUNCTION:  DESCRIBE()
/*
*****/

```

```

describe()
{
    /*****
    /*
    /*                  DISPLAY PAGE 1 OF TEXT
    /*
    *****/
    putchar(ESCAPE);          /* Clear CRT screen.
    putchar(CLEARS);

    puts(" I.  GENERAL DESCRIPTION.\n\n");
    puts("      SPEECH.COM's basic purpose is to sample an analog input\n");
    puts("      and place the quantized digital value of the sample in\n");
    puts("      extended memory.  From there, the samples may be graphically\n");
    puts("      displayed 500 at a time on the CRT screen or placed in more\n");
    puts("      permanent storage on a magnetic disk. Previously stored\n");
    puts("      samples from a magnetic disk may of course be transferred\n");

```

```

puts("      back to the system's extended memory. A short description\n");
puts("      of the eight options contained in SPEECH.COM are printed \n");
puts("      below.\n\n");
puts("  1. ANALOG-TO-DIGITAL CONVERSION.  \n\n");
puts("      Analog-to-Digital (A/D) conversion (sampling) is \n");
puts("      accomplished by the Analog Devices module DAS1128. The\n");
puts("      module is installed as Integrated Circuit chip number 61.\n");
puts("      This module has input pins for 16 (0 thru 15) separate\n");
puts("      analog inputs. Any one of the inputs may be user selected\n");
puts("      as the channel to be sampled. The default channel is\n");
puts("      channel 0. NOTE the input signal(s) must be preconditioned\n");
puts("      by the user to be between plus and minus 5 volts. During\n");

putchar(ESCAPE);          /* Display user prompt in reverse video.*/
putchar(R_VIDEO);
puts("\nPress any key to continue.");
putchar(ESCAPE);
putchar(N_VIDEO);
getchar();                /* Wait for key to be pressed.      */

/*****
/*          DISPLAY PAGE 2 OF TEXT
*****/
putchar(ESCAPE);          /* Clear CRT screen.      */
putchar(CLEAR);

puts("      program operation, the user selects the rate (in samples\n");
puts("      per second) at which the input signal will be sampled plus\n");
puts("      the number of samples to be taken and stored in extended\n");
puts("      memory. NOTE also the maximum sampling rate allowed is\n");
puts("      31,700 samples per second and the maximum number of\n");
puts("      samples storable is 163,840. (Five SSM Microcomputer static\n");
puts("      memory boards @ 32K samples -- 64K bytes -- per board.)\n");
puts("      When an A/D conversion is requested, the selected\n");
puts("      analog input is converted to 12 bit resolution digital\n");
puts("      values. The 12 bits are in 2's complement format. After\n");
puts("      each sampling of the input analog signal, first the\n");
puts("      lower 8 bits are stored in memory and then the higher\n");
puts("      4 bits are stored. The two trips to memory is due to\n");
puts("      the system's 8 bit data bus. The first memory location\n");
puts("      used for storage is always memory address 0 of the\n");
puts("      first extended memory board. The controlling devices\n");
puts("      during A/D conversion are (1) Advanced Micro Devices\n");
puts("      AM9517A Multimode Direct Memory Access (DMA) Controller,\n");
puts("      installed as Integrated Circuit chip #27, and\n");
puts("      (2) Advanced Micro Devices AM9513 System Timing\n");
puts("      Controller (STC), installed as Integrated Circuit\n");
puts("      chip #17.\n");

putchar(ESCAPE);          /* Display user prompt in reverse video.*/
putchar(R_VIDEO);
puts("\nPress any key to continue.");
putchar(ESCAPE);
putchar(N_VIDEO);
getchar();                /* Wait for key to be pressed.      */

```

```

/*****
/*          DISPLAY PAGE 3 OF TEXT          */
*****/
putchar(ESCAPE);          /* Clear CRT screen.          */
putchar(CLEAR);

puts(" 2. DIGITAL-TO-ANALOG CONVERSION.\n\n");
puts("    Digital-to-Analog (D/A) conversion is accomplished by the\n");
puts("    Analog Devices module DAC1118. The module is installed as\n");
puts("    Integrated Circuit chip #45 and uses 12 bit resolution\n");
puts("    digital values to produce a single analog signal. The 12\n");
puts("    bits are in 2's complement format and form an analog output\n");
puts("    value between plus and minus 5 volts. As with A/D sampling,\n");
puts("    two trips to memory are required to bring the 12 bits to\n");
puts("    the D/A module. The 12 bits are latched until the strobe\n");
puts("    clock pulse is received. Under software control, the system\n");
puts("    user selects the first and last samples to be converted by\n");
puts("    the module. An output delay occurs when the first sample\n");
puts("    selected is not near the beginning of an extended memory\n");
puts("    board. This is because the D/A output process only knows\n");
puts("    how to start counting bytes when starting at the beginning\n");
puts("    of an extended memory board. Therefore, an output delay is\n");
puts("    experienced as the program by-passes unrequested samples.\n");
puts("    The AM9517A DMA and AM9513 STC are the controlling devices\n");
puts("    during D/A conversion.\n");

putchar(ESCAPE);          /* Display user prompt in reverse video.*/
putchar(R_VIDEO);
puts("\nPress any key to continue.");
putchar(ESCAPE);
putchar(N_VIDEO);
getchar();                /* Wait for key to be pressed.          */

/*****
/*          DISPLAY PAGE 4 OF TEXT          */
*****/
putchar(ESCAPE);          /* Clear CRT screen.          */
putchar(CLEAR);

puts(" 3. STORE DATA ON MAGNETIC DISK.\n\n");
puts("    After performing an A/D conversion, the user may select\n");
puts("    to place the collected data samples on magnetic disk for\n");
puts("    storage. The transfer of data is always to the disk inserted\n");
puts("    in drive B. A maximum of 327,680 bytes (the capacity of all\n");
puts("    extended memory boards combined) can be placed on magnetic\n");
puts("    disk during one command. The program transfers the data\n");
puts("    samples in blocks of 2048 bytes by using a transparent data\n");
puts("    buffer. Data is first moved from extended memory to the data\n");
puts("    buffer in LO memory, and then to the magnetic disk. The\n");
puts("    data may be placed under any filename selected by the user.\n");
puts("    The default filename is DATA.ONE. The AM9517A DMA chip is\n");
puts("    used to transfer the data between HI memory (memory located\n");
puts("    above the first 64K) and LO memory (the first 64K).\n");

```

```

putchar(ESCAPE);          /* Display user prompt in reverse video.*/
putchar(R_VIDEO);
puts("\nPress any key to continue.");
putchar(ESCAPE);
putchar(N_VIDEO);
getchar();                /* Wait for key to be pressed.      */

/*****
/*      DISPLAY PAGE 5 OF TEXT
*****/
putchar(ESCAPE);          /* Clear CRT screen.      */
putchar(CLEAR);

puts(" 4. RETRIEVE DATA FROM MAGNETIC DISK.\n\n");
puts("    Data stored on magnetic disk may be placed in extended\n");
puts("    memory for use by other program options. The transfer of\n");
puts("    data samples is done in blocks of 2048 bytes by using a\n");
puts("    transparent data buffer. The data samples are first moved\n");
puts("    from the magnetic disk to a data buffer in L0 memory. The\n");
puts("    2048 byte block of data is then transferred to extended\n");
puts("    memory, beginning with byte address 0 of the first extended\n");
puts("    memory board. The AM9517A DMA chip is used to transfer the\n");
puts("    data between HI memory (memory located above the first 64K)\n");
puts("    and L0 memory (the first 64K).\n\n");

putchar(ESCAPE);          /* Display user prompt in reverse video.*/
putchar(R_VIDEO);
puts("\nPress any key to continue.");
putchar(ESCAPE);
putchar(N_VIDEO);
getchar();                /* Wait for key to be pressed.      */

/*****
/*      DISPLAY PAGE 6 OF TEXT
*****/
putchar(ESCAPE);          /* Clear CRT screen.      */
putchar(CLEAR);

puts(" 5. USE GRAPHICS TO DISPLAY DATA.\n\n");
puts("    Once an analog signal has been sampled, it can be displayed\n");
puts("    graphically on the CRT screen in blocks of 500 samples.\n");
puts("    The high resolution graphics package contained in the CRT\n");
puts("    terminal allows this to be done. The user selects the first\n");
puts("    sample of the 500 sample block to be displayed. The\n");
puts("    corresponding 1000 bytes (2 bytes per data sample) are then\n");
puts("    transferred by the AM9517A DMA chip to the transparent data\n");
puts("    buffer. These 1000 bytes are converted to the 500 y-axis\n");
puts("    values required for plotting. Once plotted a horizontal axis\n");
puts("    cursor allows selection of a beginning and ending sample\n");
puts("    for output by the D/A module. The beginning and ending sample\n");
puts("    need not be from the same 500 sample plot. Cursor movements\n");
puts("    in step of 1, 25, 100, and 500 samples are allowed.\n");
puts("    Additionally, a vertical axis (voltage line) cursor can be\n");
puts("    moved in steps of 1 and 20 pixels (20 pixels = 1 volt) to\n");
puts("    judge relative heights of the plotted data samples. All\n");
puts("    options are user prompted on the CRT screen along with the\n");
puts("    graph.\n");

```

```

    putchar(ESCAPE);          /* Display user prompt in reverse video.*/
    putchar(R_VIDEO);
    puts("\nPress any key to continue.");
    putchar(ESCAPE);
    putchar(N_VIDEO);
    getchar();                /* Wait for key to be pressed.      */

    /******
    /*          DISPLAY PAGE 7 OF TEXT
    /******
    putchar(ESCAPE);          /* Clear CRT screen.
    putchar(CLEAR);

    puts(" 6. TRANSMIT DATA TO/FROM THE ECLIPSE.\n\n");
    puts("      This option was not written as part of the thesis effort.\n");
    puts("      Space was reserved, however, in the S-100 open frame for a\n");
    puts("      TU-ART board which can be used to transfer data between the\n");
    puts("      CROMENCO S-100 microcomputer system and the NOVA/ECLISPE\n");
    puts("      minicomputer system.\n\n");
    puts(" 7. REVIEW THE INTRODUCTION.\n\n");
    puts("      This option allows the user to review the basic workings\n");
    puts("      of the program whenever the user wishes. Once the review is\n");
    puts("      completed, the program MENU is redisplayed to the user on\n");
    puts("      the CRT screen.\n\n");
    puts(" 8. EXIT THE PROGRAM.\n\n");
    puts("      This option is used whenever all other actions are done\n");
    puts("      by the user. This option displays a good-bye message to the\n");
    puts("      user and causes a control word to be set which results in\n");
    puts("      the operating system prompt (A.) being redisplayed.\n");
    putchar(ESCAPE);          /* Display user prompt in reverse video.*/
    putchar(R_VIDEO);
    puts("\nPress any key to continue.");
    putchar(ESCAPE);
    putchar(N_VIDEO);
    getchar();                /* Wait for key to be pressed.      */

    /******
    /*          DISPLAY PAGE 8 OF TEXT
    /******
    putchar(ESCAPE);          /* Clear CRT screen.
    putchar(CLEAR);

    puts("II. USER's GUIDE.\n\n");
    puts(" 1. Turn system power on.\n\n");
    puts(" 2. Place a diskette containing SPEECH.COM and the operating\n");
    puts("     system in drive A.\n\n");
    puts(" 3. Place a second diskette in drive B. The second diskette\n");
    puts("     is used for data storage and retrieval. If used for data\n");
    puts("     storage, the diskette must be able to store a number of\n");
    puts("     bytes equal to twice the number of data samples taken.\n");
    puts("     Recall that the maximum number of bytes taken by SPEECH.COM\n");
    puts("     is 327,680 bytes (163,840 samples). If used for retrieval,\n");
    puts("     the diskette must contain previously stored data.\n\n");
    puts(" 4. Boot the system by pressing the keyboard RETURN key four\n");
    puts("     times. You may alternately press the keyboard RETURN and\n");
    puts("     REPEAT keys simultaneously.\n\n");
    puts(" 5. When the system prompt (A.) appears, type: SPEECH <CR>.\n\n");

```

```

putchar(ESCAPE);          /* Display user prompt in reverse video.*/
putchar(R_VIDEO);
puts("\nPress any key to continue.");
putchar(ESCAPE);
putchar(N_VIDEO);
getchar();                /* Wait for key to be pressed.      */

/*****
/*          DISPLAY PAGE 9 OF TEXT          */
*****/
putchar(ESCAPE);          /* Clear CRT screen.      */
putchar(CLEAR);

puts(" 6. The SPEECH program is now loaded and operating. The \n");
puts("    program is user friendly and will prompt the user at all\n");
puts("    decision points.\n\n");
puts(" 7. First, the user may personalize the program by entering the\n");
puts("    user's name when asked at the beginning of the program.\n\n");
puts(" 8. Next the user will be given an opportunity to review how\n");
puts("    SPEECH.COM works. A yes or no response is solicited with\n");
puts("    the program responding accordingly.\n\n");
puts(" 9. Finally, SPEECH.COM's \"MENU\" will be displayed. The menu\n");
puts("    gives the user a choice of eight options:\n");
puts("    a. Do an ANALOG-TO-DIGITAL conversion.\n");
puts("    b. Do a DIGITAL-TO-ANALOG conversion.\n");
puts("    c. STORE data on magnetic disk.\n");
puts("    d. RETRIEVE data from magnetic disk.\n");
puts("    e. Use GRAPHICS to display data.\n");
puts("    f. TRANSMIT data to the Eclipse.\n");
puts("    (Not written as part of AFIT THESIS GE/EE/83D-38.)\n");
puts("    g. Review the INTRODUCTION.\n");
puts("    h. EXIT the program.\n");

putchar(ESCAPE);          /* Display user prompt in reverse video.*/
putchar(R_VIDEO);
puts("\nPress any key to continue.");
putchar(ESCAPE);
putchar(N_VIDEO);
getchar();                /* Wait for key to be pressed.      */

/*****
/*          DISPLAY PAGE 10 OF TEXT          */
*****/
putchar(ESCAPE);          /* Clear CRT screen.      */
putchar(CLEAR);

puts(" 10. After completing any menu option (except of course EXIT),\n");
puts("    the \"MENU\" is redisplayed to the user. Another choice\n");
puts("    can then be entered.\n\n");
puts(" 11. When the user is finished, type \"E\" and the system prompt\n");
puts("    will reappear on the CRT screen.\n\n");
puts(" 12. Remove all diskettes, turn off all system power, and the\n");
puts("    session is completed.\n");
puts("\nTHAT CONCLUDES THE INTRODUCTION TO \"SPEECH\".\n");
puts("NOW ON TO THE MENU.\n");

```

```

putchar(ESCAPE);          /* Display user prompt in reverse video.*/
putchar(R_VIDEO);
puts("\nPress any key to continue.");
putchar(ESCAPE);
putchar(N_VIDEO);
getchar();                /* Wait for key to be pressed.      */
}

```

```

/*****
/*
/*      NAME:      DEFAULTS.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 6
/*      FUNCTION:  Initializes specified GLOBAL VARIABLES to default
/*                  values.
/*      INPUTS:    NONE.
/*      OUTPUTS:   NONE.
/*      GLOBAL VARIABLES USED:  exit, is_analog, is_digital, is_mem_mem,
/*                  is_graphics, is_clearmem, channel, rate, max_rate, samples,
/*                  max_samples, begin_at, crystal, filename, limit1A, limit2A,
/*                  limit3A, limit4A, limit5A, limit6A, limit7A, limit8A.
/*      GLOBAL VARIABLES CHANGED:  See GLOBAL VARIABLES USED.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  NONE.
/*      CALLING MODULES:  main().
/*
/*      AUTHOR:  CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:  AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station.  Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
*****/
#define FALSE 0          /* Logic "false" is a zero.

```

```

/*****
/*
/*                  GLOBAL VARIABLES
/*
*****/
#include "speech.h"      /* Contains all GLOBAL VARIABLES.

```

```

/*****
/*
/*                  FUNCTION:  SET_DEFAULTS()
/*
*****/
set_defaults()
(
    exit = FALSE;
    is_analog = FALSE;
    is_digital = FALSE;
    is_mem_mem = FALSE;
    is_graphics = FALSE;
    is_clearmem = FALSE;

```

```

channel[0] = '0';          /* Default analog channel to sample */
channel[1] = '\0';         /* from is: 0. */

rate[0] = '1';            /* Default sampling rate is: 10,000. */
rate[1] = '0';            /* (Samples per second.) */
rate[2] = '0';
rate[3] = '0';
rate[4] = '0';
rate[5] = '\0';

max_rate[0] = '3';        /* Maximum sampling rate is: 31,700. */
max_rate[1] = '1';        /* (Samples per second.) */
max_rate[2] = '7';
max_rate[3] = '0';
max_rate[4] = '0';
max_rate[5] = '\0';

samples[0] = '1';         /* Default number of samples to be */
samples[1] = '6';         /* taken is: 163,840. */
samples[2] = '3';
samples[3] = '8';
samples[4] = '4';
samples[5] = '0';
samples[6] = '\0';

max_samples[0] = '1';     /* Maximum number of samples which can */
max_samples[1] = '6';     /* be stored in memory is: 163,840. */
max_samples[2] = '3';
max_samples[3] = '8';
max_samples[4] = '4';
max_samples[5] = '0';
max_samples[6] = '\0';

begin_at[0] = '1';        /* Default Digital-to-Analog samples to */
begin_at[1] = '\0';       /* be by-passed is: 1. */

crystal[0] = '4';         /* The STC crystal frequency is: */
crystal[1] = '0';         /* 4,000,010 hertz. */
crystal[2] = '0';
crystal[3] = '0';
crystal[4] = '0';
crystal[5] = '1';
crystal[6] = '0';
crystal[7] = '\0';

filename[0] = 'B';        /* Default filename is: B:DATA.ONE */
filename[1] = ':';
filename[2] = 'D';
filename[3] = 'A';
filename[4] = 'T';
filename[5] = 'A';
filename[6] = '.';
filename[7] = '0';
filename[8] = 'N';
filename[9] = 'E';
filename[10] = '\0';

```

limit1A[0] = '0';	/* The first sample on memory board #1 */
limit1A[1] = '\0';	/* is: 0. */
limit2A[0] = '3';	/* The last sample on memory board #1 */
limit2A[1] = '2';	/* is: 32,768. */
limit2A[2] = '7';	
limit2A[3] = '6';	
limit2A[4] = '8';	
limit2A[5] = '\0';	
limit3A[0] = '3';	/* The first sample on memory board #2 */
limit3A[1] = '2';	/* is: 32,769. */
limit3A[2] = '7';	
limit3A[3] = '6';	
limit3A[4] = '9';	
limit3A[5] = '\0';	
limit4A[0] = '6';	/* The last sample on memory board #2 */
limit4A[1] = '5';	/* is: 65,536. */
limit4A[2] = '5';	
limit4A[3] = '3';	
limit4A[4] = '6';	
limit4A[5] = '\0';	
limit5A[0] = '6';	/* The first sample on memory board #3 */
limit5A[1] = '5';	/* is: 65,537. */
limit5A[2] = '5';	
limit5A[3] = '3';	
limit5A[4] = '7';	
limit5A[5] = '\0';	
limit6A[0] = '9';	/* The last sample on memory board #3 */
limit6A[1] = '8';	/* is: 98,304. */
limit6A[2] = '3';	
limit6A[3] = '0';	
limit6A[4] = '4';	
limit6A[5] = '\0';	
limit7A[0] = '9';	/* The first sample on memory board #4 */
limit7A[1] = '8';	/* is: 98,305. */
limit7A[2] = '3';	
limit7A[3] = '0';	
limit7A[4] = '5';	
limit7A[5] = '\0';	
limit8A[0] = '1';	/* The last sample on memory board #4 */
limit8A[1] = '3';	/* is: 131,072. */
limit8A[2] = '1';	
limit8A[3] = '0';	
limit8A[4] = '7';	
limit8A[5] = '2';	
limit8A[6] = '\0';	

)

```

*****/
/*
/*      NAME:      MENU.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 7
/*      FUNCTION:  Prompts user with "menu of commands" available in
/*      SPEECH.  Implements user's response.
/*      INPUTS:    User entered commands from H-19 keyboard.
/*      OUTPUTS:   User prompts displayed on CRT screen.
/*      GLOBAL VARIABLES USED:  name.
/*      GLOBAL VARIABLES CHANGED:  NONE.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  analog(), digital(), store(), retrieve(),
/*      graphics(), describe(), quit().
/*      CALLING MODULES:  main()
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*      Speech Work Station.  Thesis Advisor: Major Larry Kizer.
/*
/*
*****/

```

```

*****/
/*
/*      SYMBOLIC CONSTANTS
/*
*****/
#define ESCAPE      27  /* H-19 CRT ASCII "escape" code.
#define CLEARS      69  /* H-19 CRT ASCII clear the screen code.
#define R_VIDEO     0x70 /* H-19 CRT code for enter reverse video mode.
#define N_VIDEO     0x71 /* H-19 CRT code for enter normal video mode.

```

```

*****/
/*
/*      GLOBAL VARIABLES
/*
*****/
#include "speech.h" /* Contains all GLOBAL VARIABLES.

```

```

*****/
/*
/*      FUNCTION:  MENU()
/*
*****/
menu()
{
/*
/*      LOCAL VARIABLES
/*
int choose_one; /* Holds keyboard response to getchar().*/

```

```

/*****
/*      DISPLAY "MENU OF COMMANDS" TO USER      */
*****/
putchar(ESCAPE);          /* Clear CRT screen.      */
putchar(CLEAR);

LOOP2: ;                  /* Display "menu of commands" on CRT      */
    puts("\n\t");          /*      screen.      */
    putchar(ESCAPE);
    putchar(R_VIDEO);
    printf("%s ", name);
    puts("THIS IS SPEECH'S MENU:");
    putchar(ESCAPE);
    putchar(N_VIDEO);

    puts("\n\nDo an ANALOG-to-DIGITAL conversion.....Press A.\n");
    puts("\nDo a DIGITAL-to-ANALOG conversion.....Press D.\n");
    puts("\nSTORE data on magnetic disk.....Press S.\n");
    puts("\nRETRIEVE data from magnetic disk.....Press R.\n");
    puts("\nUse GRAPHICS to display data.....Press G.\n");
    puts("\nTRANSMIT data to/from Eclipse.....Press T.\n");
    puts("\nReview the INTRODUCTION.....Press I.\n");
    puts("\nEXIT the program.....Press E.\n");

    puts("\n\n\n\n");          /* Prompt user to enter a command.      */
    putchar(ESCAPE);
    putchar(R_VIDEO);
    puts("Please enter your choice.");
    putchar(ESCAPE);
    putchar(N_VIDEO);

/*****
/*      RESPOND TO USER'S COMMAND      */
*****/
choose_one = getchar();    /* Get user's command.      */
switch(choose_one){        /* Respond to user's command.      */
    case 'A': case 'a':    /* Do an ANALOG-to-DIGITAL conversion.      */
        analog();
        break;
    case 'D': case 'd':    /* Do a DIGITAL-to-ANALOG conversion.      */
        digital();
        break;
    case 'S': case 's':    /* STORE data on magnetic disk.      */
        store();
        break;
    case 'R': case 'r':    /* RETRIEVE data from magnetic disk.      */
        retrieve();
        break;
    case 'G': case 'g':    /* Use GRAPHICS to display data.      */
        graphics();
        break;
    case 'T': case 't':    /* TRANSMIT data to Eclipse.      */
        puts("\nCase T will not be written for this thesis.\n");
        sleep(60);
        break;
}

```

```

case 'I': case 'i':      /* Review the INTRODUCTION.      */
    describe();
    break;
case 'E': case 'e':      /* EXIT the program.      */
    quit();
    break;
default:                 /* User didn't enter a valid command. */
    putchar(ESCAPE);      /* Clear CRT screen and reprompt user. */
    putchar(CLEARARS);
    printf("Sorry %s, ", name);
    puts("I didn't understand what you said.\n");
    goto LOOP2;           /* Go wait for another command.      */

```

```

    )
}

```

```

/*****
/*
/*      NAME:      QUIT.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 8
/*      FUNCTION:   Displays goodbye message to user on CRT screen and
/*                  exits program.
/*      INPUTS:     NONE.
/*      OUTPUTS:    Programmed "text" displayed on CRT screen.
/*      GLOBAL VARIABLES USED: name, exit.
/*      GLOBAL VARIABLES CHANGED: exit.
/*      FILES READ: NONE.
/*      FILES WRITTEN: NONE.
/*      MODULES CALLED: NONE.
/*      CALLING MODULES: menu(), analog(), digital(), graphics().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/834-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
*****/
#define ESCAPE      27          /* H-19 CRT ASCII "escape" code.
#define CLEARS      69          /* H-19 CRT ASCII clear the screen code.
#define BELL        7           /* H-19 CRT ASCII bell code.
#define TRUE        1           /* Logic "true" is a 1.

/*****
/*
/*                  GLOBAL VARIABLES
/*
*****/
#include "speech.h"           /* Contains all GLOBAL VARIABLES.

/*****
/*
/*                  FUNCTION:  QUIT
/*
*****/
quit()
{
    putchar(ESCAPE);          /* Clear CRT screen and display goodbye
    putchar(CLEARS);          /* message to user.

    printf("\n\n\n\nHAVE A NICE DAY %s!\n", name);
    puts("\nBYE!\n");

    putchar(BELL);            /* Ring Bell.
    exit = TRUE;              /* Enable program to return to
                              /* operating system.
}

```

```

/*****
/*
/*      NAME:      ANALOG.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 9
/*      FUNCTION:  Performs analog-to-digital conversion of analog
/*                  input. User selects "analog channel" to be sampled (1 of 16),
/*                  data "sampling rate", and total "number of samples" to be
/*                  taken.
/*      INPUTS:    (1) User entered commands from H-19 keyboard.
/*                  (2) Analog input to A/D module (DAS 1128, IC #61).
/*      OUTPUTS:   User prompts displayed on CRT screen.
/*      GLOBAL VARIABLES USED:  is_analog, chan_num.
/*      GLOBAL VARIABLES CHANGED: is_analog.
/*      FILES READ:  NONE.
/*      FILES WRITTEN: NONE.
/*      MODULES CALLED:  clearmen(), input(), dma(), timing(), quit(),
/*                      wait().
/*      CALLING MODULES:  menu().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*      SYMBOLIC CONSTANTS
/*
*****/
#define ESCAPE      27    /* H-19 CRT ASCII "escape" code.
#define CLEARS      69    /* H-19 CRT ASCII clear screen code.
#define CLEARH      0xC0  /* Address which resets hardware.
#define BELL        7     /* H-19 CRT ASCII ring bell code.
#define R_VIDEO     0x70  /* H-19 CRT code - enter reverse video mode.
#define N_VIDEO     0x71  /* H-19 CRT code - enter normal video mode.
#define STC_C       0x11  /* Address for System Timing Controller (STC)
/* chip select - control register transfer.
#define RUN         0x30  /* Arm STC register 5 -- start the CLOCK.
#define RESET       0xFF  /* STC Master Reset code.
#define A_MUX       0xA0  /* Address used to load analog input channel
/* to be sampled.
#define EXTENDED    0x70  /* Address used to load Extended Address.
#define BOARD_1     0x01  /* Address of first extended memory board.
#define A_TO_D      0x60  /* Address which toggles Analog-to-Digital
/* mode switch ON-OFF.
#define TOGGLE      0     /* TOGGLE may be any value. TOGGLE is used
/* in the 'value' position of the "C"
/* statement: outp(port,value).
#define TRUE        1     /* Logic "true" is a 1.
#define FALSE       0     /* Logic "false" is a 0.

```

```

/*****
/*                                GLOBAL VARIABLES                                */
/*****
#include "speech.h"                /* Contains all GLOBAL VARIABLES          */

/*****
/*                                FUNCTION: ANALOG()                             */
/*****
analog()
(
    /*****
    /*                                LOCAL VARIABLES                                */
    /*****
    int do_next;                    /* Holds keyboard response to getchar().*/
    char chan_in;                  /* Holds 8 bit version of analog input  */
                                /* channel to be sampled.                */

    /*****
    /*                                SET UP FOR ANALOG-TO-DIGITAL SAMPLING          */
    /*****
    is_analog = TRUE;              /* Enable selected portions of other   */
                                /* functions used by analog().          */
    putchar(ESCAPE);              /* Clear CRT screen.                   */
    putchar(CLEARH);              /* Load all Extended Memory Board bytes */
    clearmem();                  /* with zeros (00000000B).             */
    outp(CLEARH, TOGGLE);         /* RESET Thesis developed hardware.     */
    puts("\b \b");               /* NOTE: CLEARH = COH. The CRT strips   */
                                /* parity bit and prints "@" on the     */
                                /* screen. (@ = 40H). Erase "@".        */
    input();                     /* Input user's desired analog "input  */
                                /* channel", data "sampling rate", and  */
                                /* total "number of samples" to take.   */
    dma();                       /* Set DMA chip for analog-to-digital   */
                                /* sampling.                             */
    timing();                    /* Set STC chip for analog_to_digital   */
                                /* sampling.                             */
    chan_in = chan_num;          /* Form 8 bit version of analog channel */
                                /* to be sampled.                       */
    outp(A_MUX, chan_in);        /* Load analog input channel to be    */
                                /* sampled.                             */
    outp(EXTENDED, BOARD_1);     /* Load address of initial extended    */
                                /* memory board to be used.            */
    outp(A_TO_D, TOGGLE);        /* Toggle A/D mode switch to: ON.      */

```

```

/*****
/*          PERFORM ANALOG-TO-DIGITAL SAMPLING          */
*****/
/* Display user prompts on CRT screen. */

puts("\n\n\n\n\n\n\n\n\n\n");
puts("\nMAGIC's ready for ANALOG-to-DIGITAL sampling.\n\n");
putchar(ESCAPE);
putchar(R_VIDEO);
puts("  E- EXIT          M - MENU          ANY OTHER KEY - RUN  ");
putchar(ESCAPE);
putchar(N_VIDEO);

do_next = getchar();          /* Get user's command.          */
puts("\b \n");                /* Make command invisible on CRT screen.*/

switch(do_next) {              /* Respond to user's command.      */
    case 'E': case 'e':        /* EXIT program.                  */
        quit();
        goto BYE;
    case 'M': case 'm':        /* Return to MENU.                */
        goto BYE;
    default:                   /* Let user know sampling has started by*/
        /* displaying message & ringing bell. */
        puts("\nMAGIC is at work!\n");
        putchar(BELL);
}

outp(STC_C, RUN);              /* Arm STC register 5 -- the CLOCK.  */
/* BEGIN A/D SAMPLING!!      */
wait();                        /* Place CPU in tight loop until all */
/* samples are taken.        */

/*****
/*          DONE          */
*****/
/* Let user know sampling is finished by*/
/* displaying message & ringing bell. */
putchar(BELL);
puts("\nANALOG-to-DIGITAL sampling finished.\n");
sleep(60);                     /* Let user read message.          */

BYE: ;
outp(A_TO_D, TOGGLE);          /* Toggle A/D mode switch to: OFF.  */
outp(STC_C, RESET);            /* RESET STC chip.                  */
is_analog = FALSE;             /* Disable selected portions of other */
/* functions used by analog().      */
}

```

```

/*****
/*
/*      NAME:      DIGITAL.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 10
/*      FUNCTION:  Performs digital-to-analog conversion of data samples
/*                  stored in extended memory. User selects first and last sample
/*                  to be converted.
/*      INPUTS:    User entered commands from H-19 keyboard.
/*      OUTPUTS:   (1) User prompts displayed on CRT screen.
/*                  (2) Analog output signal from D/A module (DAC 1118,
/*                      IC #45).
/*      GLOBAL VARIABLES USED:  is_digital, is_graphics, limit4A, start.
/*      GLOBAL VARIABLES CHANGED: is_digital.
/*      FILES READ:  NONE.
/*      FILES WRITTEN: NONE.
/*      MODULES CALLED: input(), dma(), timing(), quit(), wait().
/*      CALLING MODULES: menu(), graphics().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*      SYMBOLIC CONSTANTS
/*
*****/
#define ESCAPE      27    /* H-19 CRT ASCII "escape" code.
#define CLEARH      69    /* H-19 CRT ASCII clear screen code.
#define CLEARH      0xCO  /* Address which resets hardware.
#define BELL        7     /* H-19 CRT ASCII ring bell code.
#define R_VIDEO     0x70  /* H-19 CRT code - enter reverse video mode.
#define N_VIDEO     0x71  /* H-19 CRT code _enter normal video mode.
#define STC_C       0x11  /* Address System Timing Controller (STC)
/* chip select - control register transfer.
#define RUN         0x30  /* Arm STC register 5 -- start the CLOCK.
#define RESET       0xFF  /* STC Master Reset code.
#define EXTENDED    0x70  /* Address used to load Extended Address.
#define D_TO_A      0xF0  /* Address which toggles Digital-to-Analog
/* mode switch ON-OFF.
#define TOGGLE      0     /* TOGGLE may be any value. TOGGLE is used
/* in the 'value' position of the "C"
/* statement: outp(port,value).
#define TRUE        1     /* Logic "true" is a 1.
#define FALSE       0     /* Logic "false" is a 0.

/*****
/*
/*      GLOBAL VARIABLES
/*
*****/
#include "speech.h"      /* Contains all GLOBAL VARIABLES.

```

```

/*****
/*
/*          FUNCTION:  DIGITAL()
/*
/*
/*****
digital()
{
    /*****
    /*          LOCAL VARIABLES
    /*
    /*****
    int do_next;                /* Holds keyboard response to getchar().*/
    char board_num[4];          /* Holds address of extended memory */
                                /* board containing first "data */
                                /* sample" to be converted from */
                                /* digital-to-analog. */
    char offset[4];             /* Holds LONG (32 bit) integer used to */
                                /* calculate board_num. */
    char one[4];                /* Holds LONG (32 bit) version of 1. */
    char two[4];                /* Holds LONG (32 bit) version of 2. */

    /*****
    /*          SET UP FOR DIGITAL-TO-ANALOG CONVERSION
    /*
    /*****
    is_digital = TRUE;          /* Enable selected portions of other */
                                /* functions used by digital(). */
    if(is_graphics == TRUE)     /* Don't get inputs when in graphics(), */
        ;                       /* because they are already selected! */
    else {
        putchar(ESCAPE);        /* Clear CRT screen. */
        putchar(CLEARH);
        input();                /* Input user's desired "first and last" */
                                /* sample to be converted, and desired */
                                /* "sampling rate". */
    }
    outp(CLEARH,TOGGLE);        /* RESET Thesis developed hardware. */
    puts("\b\b");              /* NOTE: CLEARH = COH. The CRT strips */
                                /* parity bit and prints "@" on the */
                                /* screen. (@ = 40H). Erase "@". */
    dma();                      /* Set DMA chip for digital-to-analog */
                                /* conversion. */
    timing();                   /* Set STC chip for digital-to-analog */
                                /* conversion. */
    itol(one,1);                /* Convert 1 into a 32 bit integer. */
    itol(two,2);                /* Convert 2 into a 32 bit integer. */
    atol(offset,limit4A);       /* Convert limit4A to a 32 bit integer. */

    lmul(board_num,start,two);   /* Form address of extended */
    lsub(board_num,board_num,one); /* memory board containing */
    ladd(board_num,board_num,offset); /* first "data sample" to be */
                                /* digital-to-analog converted. */
    outp(EXTENDED,board_num[1]); /* Load address of initial extended */
                                /* memory board to be used. */
    outp(D_TO_A, TOGGLE);       /* Toggle D/A mode switch to: ON. */
}

```

```

/*****
/*          PERFORM DIGITAL-TO-ANALOG CONVERSION          */
/*****
if(is_graphics == TRUE)      /* Don't display prompts when in    */
    ;                        /*  graphics()!! Just do conversion. */

else {                        /* Display user prompts on CRT screen. */
    puts("\n\n\n\n\n\n\n\n\n\n");
    puts("\nMAGIC's ready for DIGITAL-to-ANALOG conversion.\n\n");
    putchar(ESCAPE);
    putchar(R_VIDEO);
    puts("  E - EXIT          M - MENU          ANY OTHER KEY - RUN  ");
    putchar(ESCAPE);
    putchar(N_VIDEO);

    do_next = getchar();      /* Get user's command.          */
    puts("\b \n");           /* Make command invisible on CRT screen.*/

    switch(do_next) {         /* Respond to user's command.    */
        case 'E': case 'e':  /* EXIT program.                */
            quit();
            goto BYE;
        case 'M': case 'm':  /* Return to MENU.              */
            goto BYE;
        default:              /* Let user know conversion has started */
            /* by displaying message & ringing bell.*/
            puts("\nMAGIC is at work!\n");
            putchar(BELL);
    }
}

outp(STC_C, RUN);            /* Arm STC register 5 -- the CLOCK. */
/* BEGIN D/A CONVERSION!!    */
wait();                      /* Place CPU in tight loop until all */
/* conversions are made.      */

/*****
/*          DONE          */
/*****
if(is_graphics == TRUE)      /* Don't display message when in    */
    ;                        /*  graphics().                    */

else {                        /* Let user know conversion is finished */
    putchar(BELL);           /* by displaying message & ringing bell.*/
    puts("\nDIGITAL-to-ANALOG conversion finished.\n");
    sleep(60);               /* Let user read message.          */
}

BYE: ;
outp(D_TO_A, TOGGLE);        /* Toggle D/A mode switch to: OFF.   */
outp(STC_C, RESET);          /* RESET STC chip.                   */
is_digital = FALSE;          /* Disable selected portions of other */
/* functions used by digital().      */
}

```

```

/*****
/*
/*      NAME:      STORE.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 11
/*      FUNCTION:  Transfer user specified number of data samples from
/*                  extended memory to magnetic disk. Transfer is to user
/*                  specified "filename". Default "filename" is: B:DATA.ONE.
/*                  Transfer is done 2048 bytes at a time (thru mem_buffer).
/*      INPUTS:    User entered commands from H-19 keyboard.
/*      OUTPUTS:   User prompts displayed on CRT screen.
/*      GLOBAL VARIABLES USED:  finish, filename, is_mem_mem, from, to,
/*                              mem_buffer.
/*      GLOBAL VARIABLES CHANGED:  filename, is_mem_mem, from, to,
/*                              mem_buffer.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  "Filename" entered by user from H-19 keyboard.
/*                  Default "filename" is: B:DATA.ONE.
/*      MODULES CALLED:  dma().
/*      CALLING MODULES:  menu().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
*****/
#define ESCAPE      27 /* H-19 CRT ASCII "escape" code.
#define CLEARS      69 /* H-19 CRT ASCII clear screen code.
#define BACKSPACE  0x08 /* ASCII hex code for "backspace" key.
#define DELETE     0x7F /* ASCII hex code for "delete" key.
#define ERROR      -1 /* General "is an error" return value.
#define TRUE        1 /* Logic "true" is a 1.
#define FALSE       0 /* Logic "false" is a 0.
#define TOGGLE      0 /* TOGGLE may be any value. TOGGLE is used in
/*                  the 'value' position of the "C" statement:
/*                  outp(port,value).
#define CPU_MEM     0xD0 /* Address which toggles between "CPU or DMA"
/*                  actions and "MEMORY-TO-MEMORY" transfers.
#define EXTENDED    0x70 /* Address used to load Extended Address.
#define REQUEST     0x99 /* Address of software DREQ Request Register.
#define MEM_XFER    0x04 /* Software DMA CHAN-0 request, i.e. start
/*                  memory-to-memory transfer.

/*****
/*
/*                  GLOBAL VARIABLES
*****/
#include "speech.h" /* Contains all GLOBAL VARIABLES.

```



```

if((c = getchar()) != '\n') {
    filename[2] = c;
    for(i = 3; (c = getchar()) != '\n' && i < 17; ++i) {
        if(c == BACKSPACE || c == DELETE)
            --i;
        else
            filename[i] = c;
    }
    filename[i] = '\0';
}

/*****
/*      STORE DATA ON MAGNETIC DISK      */
*****/
outp(CPU_MEM,TOGGLE);          /* Set hardware for "Memory-to-Memory" */
                                /* data transfer. */
printf("\nMAGIC is now STORING data in file %s\n",filename);

fd = creat(filename);          /* Create FILE DESCRIPTOR for filename. */
if(fd == ERROR) {              /* Check for and respond to any errors. */
    printf("\nERROR: Can't creat %s\n",filename);
    puts("\nPress any key to continue.\n");
    getchar();
    goto BYE;
}

is_mem_mem = TRUE;            /* Enable selected portions of other */
                                /* functions used by store(). */
from[3] = 0x00;                /* Initial memory address where data is */
from[2] = 0x00;                /* found. Used by dma(). */
from[1] = 0x01;
from[0] = 0x00;

ptr_num = &mem_buffer[0];     /* Find memory address of buffer array. */
holder = ptr_num;

to[3] = ptr_num;               /* Initial memory address of buffer used */
to[2] = holder >> 8;           /* to transfer 2048 byte blocks of */
to[1] = 0x00;                  /* data to magnetic disk storage. */
to[0] = 0x00;                  /* Used by dma(). */

for(i = 1; i <= done; ++i) { /* Transfer data. */
    outp(EXTENDED,from[1]);    /* Load address of 2048 byte block of */
                                /* data to be stored. */
    dma();                     /* Set DMA for data transfer. */
    outp(REQUEST,MEM_XFER);    /* Begin 2048 byte "Memory-to-Memory" */
                                /* data transfer. */
    ladd(from,from,buffer_size); /* Address of next 2048 byte block */
                                /* of data to be stored on disk. */

                                /* Transfer data from buffer array to */
                                /* magnetic disk. Check for and */
                                /* respond to any errors. */
}

```

```

        if(write(fd,mem_buffer,16) == ERROR) {
            puts("\nERROR: Probably out of disk space.\n");
            puts("\nPress any key to continue.\n");
            getchar();
            goto BYE;
        }
    }
    puts("\n\nTRANSFER COMPLETED!!!\n"); /* Let user know transfer is */
    sleep(60);                          /* completed. */

    /******
    /*                                DONE                                */
    /******
BYE: ;
    outp(CPU_MEM,TOGGLE); /* Transfer done. Set hardware for */
                          /* "CPU or DMA" operation. */
    is_mem_mem = FALSE;  /* Disable selected portions of other */
                          /* functions used by store(). */
    close(fd);           /* Close FILE that was created. */
}

```

```

/*****
/*
/*      NAME:      RETRIEVE.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 12
/*      FUNCTION:  Clears extended memory. Then transfers data from
/*                  magnetic disk to extended memory. Transfer is from user
/*                  specified "filename". Default "filename" is: B:DATA.ONE.
/*                  Transfer is done 2048 bytes at a time (thru mem_buffer) as
/*                  long as data remains in the disk file.
/*      INPUTS:    User entered commands from H-19 keyboard.
/*      OUTPUTS:   User prompts displayed on CRT screen.
/*      GLOBAL VARIABLES USED:  filename, is_mem_mem, to, from,
/*                               mem_buffer.
/*      GLOBAL VARIABLES CHANGED: filename, is_mem_mem, to, from,
/*                               mem_buffer.
/*      FILES_READ: "Filename" entered by user from H-19 keyboard.
/*                  Default "filename" is: B:DATA.ONE.
/*      FILES WRITTEN: NONE.
/*      MODULES CALLED: dma(), clearmem().
/*      CALLING MODULES: menu().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
*****/
#define ESCAPE      27 /* H-19 CRT ASCII "escape" code.
#define CLEARS      69 /* H-19 CRT ASCII clear screen code.
#define BACKSPACE  0x08 /* ASCII hex code for "Backspace" key.
#define DELETE     0x7F /* ASCII hex code for "Delete" key.
#define ERROR      -1 /* General "is an error" return value.
#define TO_READ    0 /* Used to open a disk file for reading.
#define ZERO       0 /* Makes program easier to read.
#define TRUE       1 /* Logic "true" is a 1.
#define FALSE      0 /* Logic "false" is a 0.
#define TOGGLE     0 /* TOGGLE may be any value. TOGGLE is used in
/*                  the 'value' position of the "C" statement:
/*                  outp(port,value).
#define HI_LO      0xB0 /* Address which toggles memory-to-memory data
/*                  transfer between "HI-memory to LO-memory"
/*                  mode and "LO-memory to HI-memory" mode.
#define CPU_MEM     0xD0 /* Address which toggles between "CPU or DMA"
/*                  actions and "MEMORY-TO-MEMORY" transfers.
#define EXTENDED    0x70 /* Address used to load Extended Address.
#define REQUEST     0x99 /* Address of software DREQ Request Register.
#define MEM_XFER    0x04 /* Software DMA CHAN-0 request, i.e. start
/*                  memory-to-memory transfer.

```



```

/*****
/*          RETRIEVE DATA FROM MAGNETIC DISK          */
*****/
clearmem();          /* Load all Extended Memory bytes with */
                    /* zeros (00000000B). */
outp(CPU_MEM,TOGGLE); /* Set hardware for "Memory-to-Memory" */
                    /* data transfer. */
outp(HI_LO,TOGGLE);  /* Set hardware for "LO-memory to HI- */
                    /* Memory" data transfer. */

printf("\nMAGIC is RETRIEVING data from file %s\n",filename);

fd = open(filename,TO_READ); /* Create FILE DESCRIPTOR for filename. */
if(fd == ERROR) {           /* Check for and respond to any errors. */
    printf("\nERROR: Can't open %s\n",filename);
    puts("\nPress any key to continue.\n");
    getchar();
    goto BYE;
}

is_mem_mem = TRUE;          /* Enable selected portions of other */
                    /* functions used by retrieve(). */
to[3] = 0x00;               /* Initial extended memory address where */
to[2] = 0x00;               /* data is to be moved. Used by dma(). */
to[1] = 0x01;
to[0] = 0x00;

ptr_num = &mem_buffer[0];  /* Find memory address of buffer array. */
holder = ptr_num;

from[3] = ptr_num;          /* Initial memory address of buffer */
from[2] = holder >> 8;      /* used to transfer 2048 byte blocks */
from[1] = 0x00;             /* of data to extended memory. Used */
from[0] = 0x00;             /* by dma(). */

itol(buffer_size,2048);     /* Convert 2048 into a 32 bit integer. */
                    /* Transfer data from magnetic disk to */
                    /* buffer. Check for and respond to */
                    /* any errors. */

while((check = read(fd,mem_buffer,16)) != ZERO) {
    if(check == ERROR) {
        printf("\nERROR: Can't read %s\n",filename);
        puts("\nPress any key to continue.\n");
        getchar();
        goto BYE;
    }

    outp(EXTENDED,to[1]);    /* Load address where 2048 byte block */
                    /* of data is to be stored. */
    dma();                  /* Set DMA for data transfer. */
    outp(REQUEST,MEM_XFER);  /* Begin 2048 byte Memory-to-Memory */
                    /* transfer. */
    ladd(to,to,buffer_size); /* Form address where next 2048 byte */
                    /* block of data is to be stored. */
}

```

```

/*****
/*                               DONE                               */
/*****
puts("\n\nTRANSFER COMPLETED!!!"); /* Let user know transfer of */
sleep(60);                          /* data is completed.      */
BYE: ;
close(fd);                          /* Close FILE that was opened. */
outp(HI_LO,TOGGLE);                 /* Set hardware for "HI-memory to LO- */
/* Memory" data transfer.          */
outp(CPU_MEM,TOGGLE);               /* Set hardware for "CPU or DMA" mode */
/* of operation.                   */
is_mem_mem = FALSE;                 /* Disable selected portions of other */
/* functions used by retrieve().    */
)

```

```

/*****
/*
/*      NAME:      GRAPHICS.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 13
/*      FUNCTION:  Graphically displays 500 data samples on the CRT
/*                  screen. Displays user prompts. Provides right & left CURSOR
/*                  movement. Provides up & down VOLT LINE movement. Provides
/*                  selecting starting and finishing "data sample numbers" for
/*                  use during digital-to-analog output. Provides selecting
/*                  digital-to-analog output.
/*      INPUTS:    User entered commands from H-19 keyboard.
/*      OUTPUTS:   User prompts and graph of 500 data samples displayed
/*                  on CRT screen.
/*      GLOBAL VARIABLES USED:  is_graphics, cursor, begin_at, x_axis,
/*                  a_temp, l_temp, l_cursor, start, samples, finish.
/*      GLOBAL VARIABLES CHANGED: is_graphics, cursor, begin_at, x_axis,
/*                  a_temp, l_temp, l_cursor, start, samples, finish, input1,
/*                  input2.
/*      FILES READ:  NONE.
/*      FILES WRITTEN: NONE.
/*      MODULES CALLED: input(), plot(), quit(), right(), left(),
/*                  voltline(), digital().
/*      CALLING MODULES: menu().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
*****/
#define  ESCAPE      27 /* H-19 CRT ASCII "escape" code.
#define  CLEARS      69 /* H-19 CRT ASCII clear screen code.
#define  R_VIDEO     0x70 /* H-19 CRT code - enter reverse video mode.
#define  N_VIDEO     0x71 /* H-19 CRT code - enter normal video mode.
#define  BACKSPACE   0x08 /* ASCII hex code for "backspace" key.
#define  DELETE      0x7F /* ASCII hex code for "delete" key.
#define  TRUE        1 /* Logic "true" is a one.
#define  FALSE       0 /* Logic "false" is a zero.

```

```

/*****
/*
/*                  GLOBAL VARIABLES
/*
*****/
#include "speech.h" /* Contains all GLOBAL VARIABLES.

```

```

/*****
/*
/*          FUNCTION:  GRAPHICS()
/*
/*
/*****
graphics()
{
    /*****
    /*          LOCAL VARIABLES
    /*
    /*****
    int do_next;          /* Holds keyboard response to getchar().*/

    char one[4];          /* Holds LONG (32 bit) version of 1.    */
    char two[4];          /* Holds LONG (32 bit) version of 2.    */
    char hundred[4];      /* Holds LONG (32 bit) version of 100. */

    /*****
    /*          INITIALIZE VARIABLES
    /*
    /*****
    is_graphics = TRUE;   /* Enable selected portions of other
    /*          functions used by graphics().
    cursor = 251;         /* Vertical CURSOR initially placed at
    /*          sample #250 (pixel #251) of 500
    /*          sample plot.
    itol(one,1);          /* Convert 1 into a 32 bit integer.
    itol(two,2);          /* Convert 2 into a 32 bit integer.
    itol(hundred,100);    /* Convert 100 into a 32 bit integer.

    /*****
    /*          INPUT "SAMPLING RATE" AND "FIRST SAMPLE" TO BE PLOTTED
    /*
    /*****
    putchar(ESCAPE);      /* Clear CRT screen.
    putchar(CLEAR);
    input();              /* Display introduction to Graphics().
    /*          Input user's desired "sampling rate"
    /*          and "first sample" to be plotted.
    atol(x_axis,begin_at); /* Form 32 bit version of "sample #"
    /*          to be displayed in first column
    /*          (pixel #2) of graph.

    /*****
    /*          DRAW GRAPH
    /*
    /*****
    putchar(ESCAPE);      /* Clear CRT screen.
    putchar(CLEAR);
    plot();               /* Plot first 500 samples.
    puts("\0331,");        /* Enter Graphics Mode.
    puts("N,170,");        /* Primary Line Style:  DASHED LINE.
    puts("I,2,");          /* Line Type:  COMPLEMENT.
    puts("P,251,246,L,251,47,"); /* Draw vertical CURSOR.
    puts("E");             /* Exit Graphics Mode.

```

```

/*****
/*                                LABEL VERTICAL AXIS                                */
*****/
puts("      5\n\n");
puts("      4\n\n");
puts("      3\n\n");
puts("      2\n\n");
puts("      1\n\n");
puts("      0\n\n");
puts("     -1\n\n");
puts("     -2\n\n");
puts("     -3\n\n");
puts("     -4\n\n");
puts("     -5\n\n");

/*****
/*                                LABEL HORIZONTAL AXIS                                */
*****/
printf("      %-6s", ltoa(a_temp,x_axis));
lsub(l_temp,x_axis,one);
printf("%11s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%12s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%13s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%12s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%13s\n", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));

/*****
/*                                DISPLAY PROMPTS TO USER                                */
*****/
putchar(ESCAPE);          /* Display "CURSOR =" on CRT screen in */
putchar(R_VIDEO);         /* reverse video. Display current */
puts(" CURSOR =");        /* CURSOR value on CRT screen in */
putchar(ESCAPE);          /* normal video. */
putchar(N_VIDEO);
itol(l_cursor,cursor);
ladd(l_temp,x_axis,l_cursor);
printf(" %-11s",ltoa(a_temp,lsub(l_temp,l_temp,two)));

putchar(ESCAPE);          /* Display "START =" on CRT screen in */
putchar(R_VIDEO);         /* reverse video. Display current */
puts("START =");          /* START value on CRT screen in */
putchar(ESCAPE);          /* normal video. */
putchar(N_VIDEO);
printf(" %-11s",ltoa(a_temp,atol(start,begin_at)));

putchar(ESCAPE);          /* Display "FINISH =" on CRT screen in */
putchar(R_VIDEO);         /* reverse video. Display current */
puts("FINISH =");         /* FINISH value on CRT screen in */
putchar(ESCAPE);          /* normal video. */
putchar(N_VIDEO);
printf(" %-11s",ltoa(a_temp,atol(finish,samples)));

```

```

putchar(ESCAPE);          /* Display remaining user prompts on */
putchar(R_VIDEO);         /* CRT screen in reverse video. */
puts("\n");
puts(" E - EXIT      M - MENU      S - START      F - FINISH ");
puts("A - ANALOG OUT ");
puts("\033x1");           /* Enable 25th line. */
puts("\033Y8 ");          /* Go to Line 25, Column 1. */
puts(" R[1], T[25], Y[100], U[500] - RIGHT ");
puts("L[1], K[25], J[100], H[500] - LEFT");
puts("\033Y ");           /* Go to Line 1, Column 1. */
puts("UP: \nQ[1] \nW[20]\n\nDOWN:\nZ[1] \nX[20]");
putchar(ESCAPE);
putchar(N_VIDEO);

/***** RESPOND TO USER'S COMMAND *****/
/* RESPOND TO USER'S COMMAND */
/*****

E2:puts("\033Y6o");        /* Go to line 23, Column 80. Wait there */
                           /* for user's next command. */
do_next = getchar();       /* Get user's command. */
puts(" ");                /* Erase command from CRT screen. */

switch(do_next) {          /* Respond to user's command. */
    case 'E': case 'e':    /* EXIT the program. Erase graph. */
        quit();
        is_graphics = FALSE; /* Disable selected portions of other */
                           /* functions used by graphics. */
        puts("\033l,D,5,E"); /* ALPHA: on, GRAPHICS: off, ERASE: on */
        sleep(10);         /* Wait for graphics chip to stabilize. */
        goto BYE;
    case 'M': case 'm':    /* Return to MENU. Erase graph. */
        is_graphics = FALSE; /* Disable selected portions of other */
                           /* functions used by graphics. */
        puts("\033l,D,5,E"); /* ALPHA: on, GRAPHICS: off, ERASE: on */
        sleep(10);         /* Wait for graphics chip to stabilize. */
        goto BYE;
    case 'R': case 'r':    /* Move CURSOR RIGHT 1 sample. */
        input1 = 501;
        input2 = 1;
        right();
        break;
    case 'T': case 't':    /* Move CURSOR RIGHT 25 samples. */
        input1 = 477;
        input2 = 25;
        right();
        break;
    case 'Y': case 'y':    /* Move CURSOR RIGHT 100 samples. */
        input1 = 402;
        input2 = 100;
        right();
        break;
}

```

```

case 'U': case 'u':          /* Move CURSOR RIGHT 500 samples.      */
    input1 = 2;              /* Really only goes to first sample of */
    input2 = 500;            /* next 500 sample plot.              */
    right();
    break;
case 'L': case 'l':          /* Move CURSOR LEFT 1 sample.          */
    input1 = 2;
    input2 = 1;
    left();
    break;
case 'K': case 'k':          /* Move CURSOR LEFT 25 samples.        */
    input1 = 26;
    input2 = 25;
    left();
    break;
case 'J': case 'j':          /* Move CURSOR LEFT 100 samples.       */
    input1 = 101;
    input2 = 100;
    left();
    break;
case 'H': case 'h':          /* Move CURSOR LEFT 500 samples.       */
    input1 = 501;            /* Really only goes to last sample of */
    input2 = 500;            /* next 500 sample plot.              */
    left();
    break;
case 'Q': case 'q':          /* Move VOLT LINE UP 1 pixel.          */
    input1 = 1;
    voltline();
    break;
case 'W': case 'w':          /* Move VOLT LINE UP 20 pixels (1 volt).*/
    input1 = 20;
    voltline();
    break;
case 'Z': case 'z':          /* Move VOLT LINE DOWN 1 pixel.        */
    input1 = -1;
    voltline();
    break;
case 'X': case 'x':          /* Move VOLT LINE DOWN 20 pixels.      */
    input1 = -20;            /* (1 volt).                          */
    voltline();
    break;
case 'S': case 's':          /* Update STARTING "sample #" for      */
                                /* digital-to-analog output operation. */
    ladd(1_temp,x_axis,itol(1_cursor,cursor));
    ltoa(begin_at,lsub(1_temp,1_temp,two));
    puts("\033Y6<");          /* Line 23, Column 29. Update START.  */
    printf(" %-11s",ltoa(a_temp,atol(start,begin_at)));
    break;
case 'F': case 'f':          /* Update FINISHING "sample #" for     */
                                /* digital-to-analog output operation. */
    ladd(1_temp,x_axis,itol(1_cursor,cursor));
    ltoa(samples,lsub(1_temp,1_temp,two));
    puts("\033Y6P");          /* Line 23, Column 49. Update FINISH. */
    printf(" %-11s",ltoa(a_temp,atol(finish,samples)));
    break;

```

AD-A155 465

DEVELOPMENT OF A DEDICATED SPEECH WORK STATION(U) AIR  
FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF  
ENGINEERING W H LIEBER DEC 84 AFIT/GE/EE/84D-71

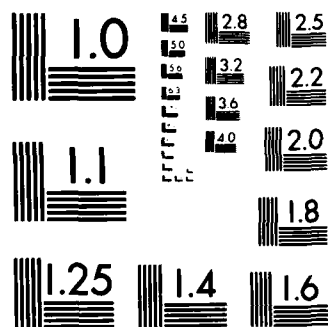
2/3

UNCLASSIFIED

F/G 9/2

NL

15



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS 1963 A

```

        case 'A': case 'a':      /* Provide a digital-to-analog output. */
            digital();
            break;
    )
    goto E2;                      /* Go wait for another user command. */

    /*****
    /*                               DONE                               */
    *****/
    BYE: ;
    puts("\033y1");              /* Disable 25th line. */
    )

```

```

/*****
/*
/*      NAME:      PLOT.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 14
/*      FUNCTION:  Draws horizontal axis, vertical axis, and zero volt
/*                  line of graph. Transfers the 500 data samples (1000 bytes) to
/*                  be displayed into mem_buffer. Calculates the y-axis values of
/*                  the 500 data samples. Plots the 500 samples on the graph.
/*                  Returns to the calling function.
/*      INPUTS:    NONE.
/*      OUTPUTS:   Displays graph of 500 data samples on CRT screen.
/*      GLOBAL VARIABLES USED:  voltline, x_axis, from, mem_buffer, to,
/*                               l_temp.
/*      GLOBAL VARIABLES CHANGED:  voltline, from, mem_buffer, to,
/*                               l_temp.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  dma().
/*      CALLING MODULES:  graphics(), right(), left().
/*
/*      AUTHOR:  CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:  AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*      SYMBOLIC CONSTANTS
/*
*****/
#define POSITIVE      0 /* Positive binary numbers begin with 0.
#define NEGATIVE      1 /* Negative binary numbers begin with 1.
#define MASK          0x08 /* Used to determine the "sign" of a SAMPLE.
#define TOGGLE        0 /* TOGGLE may be any value. TOGGLE is used in
/* the 'value' position of the "C" statement:
/* outp(port,value).
#define CPU_MEM       0xD0 /* Address which toggles between "CPU or DMA"
/* actions and "MEMORY-TO-MEMORY" transfers.
#define EXTENDED      0x70 /* Address used to load EXTENDED ADDRESS.
#define REQUEST        0x99 /* Address of software DREQ Request Register.
#define MEM_XFER       0x04 /* Software DMA CHAN-0 request, i.e. start a
/* memory-to-memory transfer.

```

```

/*****
/*
/*      GLOBAL VARIABLES
/*
*****/
#include "speech.h" /* Contains all GLOBAL VARIABLES.

```

```

/*****
/*
/*          FUNCTION: PLOT()
/*
/*
/*****
plot()
{
    /*****
    /*          LOCAL VARIABLES
    /*
    /*****
    int x;                /* Index variable used in x-axis "for"
                          /* loops.
    int y;                /* Index variable used in Y-axis "for"
                          /* loops.
    int holder;           /* Holds address found by *ptr_num.
                          /* Binary operations can be done on a
                          /* integer value, but not a pointer.
    int y_value[500];     /* Holds y-axis values (voltage levels)
                          /* to be plotted.

    unsigned complement;  /* Used to find 2's complement of a
                          /* negative data sample.

    char *ptr_num;        /* Used to find first address of
                          /* mem_buffer array.
    char sign;            /* Holds 'sign' of a data sample.
    char hundred[4];      /* Holds LONG (32 bit) version of 100.
    char num_levels[4];   /* Holds LONG (32 bit) version of number
                          /* of quantization levels used by A/D
                          /* in range 0 to 5 volts.

    /*****
    /*          DRAW HORIZONTAL AXIS, VERTICAL AXIS, & ZERO VOLT LINE
    /*
    /*****
    puts("\033l,");        /* Enter Graphics Mode.
    puts("D,7,");          /* ALPHA: on, GRAPICS: on, ERASE: on
    puts("I,0,");          /* Line Type: ON.
    puts("N,255,");        /* Primary Line Style: SOLID.
                          /* NOTE: 255 = (11111111)B
    puts("O,0,");          /* Secondary Line Style: BLANK.
    sleep(6);              /* NOTE: A delay is needed to allow
                          /* the graphics buffer to empty
                          /* SET UP commands prior to
                          /* getting movement commands.

    for(y = 66; y <= 246; y = y + 20) /* Draw vertical axis ticks.
        printf("P,0,%d,L,4,%d,",y,y);
    for(y = 56; y <= 236; y = y + 20)
        printf("P,1,%d,L,3,%d,",y,y);

    for(x = 101; x <= 501; x = x + 100) /* Draw horizontal axis ticks.
        printf("P,%d,44,L,%d,48,",x,x);
    for(x = 26; x <= 481; x = x + 25)
        printf("P,%d,45,L,%d,47,",x,x);

```

```

puts("P,2,245,L,2,46,");      /* Draw vertical axis.          */
puts("L,500,46,");            /* Draw horizontal axis.        */
puts("N,170,");               /* Primary Line Style: DASHED LINE. */
                               /* NOTE: 170 = (10101010)B      */
volt_line = 146;              /* Draw ZERO volt line.         */
printf("P,4,%d,L,501,%d,",volt_line,volt_line);

```

/\* NOTE1: REMOVE COMMENT DESIGNATOR AFTER DMA MEMORY-TO-MEMORY TRANSFER  
IS WORKING. MUST "CC1" PLOT.C, "CLIB" PLOT.CRL INTO HOLD2.CRL,  
AND "CLINK" SPEECH SPEECH1 SPEECH2 TO FORM NEW SPEECH.COM FILE.

```

/*****
/*          BRING DOWN NEXT 1000 BYTES          */
*****/
from[3] = x_axis[3];          /* Initial memory address where data is */
from[2] = x_axis[2];          /* to be found.                        */
from[1] = x_axis[1];
from[0] = x_axis[0];

ptr_num = &mem_buffer[0];    /* Find memory address of buffer array. */
holder = ptr_num;

to[3] = ptr_num;              /* Initial memory address where data is */
to[2] = holder >> 8;          /* to be moved.                        */
to[1] = 0x00;
to[0] = 0x00;

outp(CPU_MEM,TOGGLE);         /* Set hardware for Memory-to-Memory */
                               /* data transfer.                    */
outp(EXTENDED,from[1]);       /* Load extended address.            */
dma();                         /* Set DMA for data transfer.          */
outp(REQUEST,MEM_XFER);       /* Begin the Memory-to-Memory transfer. */
outp(CPU_MEM,TOGGLE);         /* Transfer done. Set hardware for CPU */
                               /* or DMA operations.                 */

/*****
/*          CREATE Y_VALUES FOR NEXT 500 SAMPLES          */
*****/
itol(hundred,100);            /* Convert 100 into a 32 bit integer. */
itol(num_levels,2047);        /* Convert 2047 into a 32 bit integer. */

for(x = 1; x <= 500; ++x) {
    sign = (mem_buffer[2 * x - 1] & MASK);    /* Find sign bit for */
    sign = sign >> 3;                        /* data sample.      */

    l_temp[0] = 0x00;                        /* Convert data sample */
    l_temp[1] = 0x00;                        /* into a 32 bit      */
    l_temp[2] = mem_buffer[2 * x - 1];       /* integer.            */
    l_temp[3] = mem_buffer[2 * x - 2];

    if(sign == POSITIVE) {                  /* If sign is positive, */
        lmul(l_temp,l_temp,hundred);        /* form y-axis voltage*/
        ldiv(l_temp,l_temp,num_levels);     /* level for plotting.*/
        y_value[x - 1] = (146 + ltou(l_temp));
    }
}

```

```

    } else {
        complement = (1 + (~(ltou(1_temp)))); /* Sign is negative. */
        utol(1_temp, complement); /* Take 2's complement */
        lmul(1_temp, 1_temp, hundred); /* and form y-axis */
        ldiv(1_temp, 1_temp, num_levels); /* voltage level for */
        y_value[x - 1] = (146 - ltou(1_temp)); /* plotting. */
    }
}

```

NOTE2: SEE NOTE1. REMOVE THE FOLLOWING COMMENT DESIGNATOR AND THE  
 "FOR TEST PURPOSES: CREATE Y\_VALUES" CODE.

\*/

```

/*****
/*          FOR TEST PURPOSES:  CREATE Y_VALUES          */
/*****
for(x = 0; x <= 60; ++x) /* This data will form a stair step */
    y_value[x] = 46; /* plot on the graph. */
for(x = 61; x <= 120; ++x)
    y_value[x] = 76;
for(x = 121; x <= 180; ++x)
    y_value[x] = 106;
for(x = 181; x <= 240; ++x)
    y_value[x] = 136;
for(x = 241; x <= 300; ++x)
    y_value[x] = 166;
for(x = 301; x <= 360; ++x)
    y_value[x] = 196;
for(x = 361; x <= 420; ++x)
    y_value[x] = 216;
for(x = 421; x <= 480; ++x)
    y_value[x] = 246;
for(x = 481; x <= 499; ++x)
    y_value[x] = 56;

/*****
/*          PLOT NEXT 500 SAMPLES          */
/*****
puts("N,255,"); /* Primary Line Style: SOLID. */
printf("P,2,%d", y_value[0]); /* Plot samples. */
for(x = 3; x <= 501; ++x)
    printf("L,%d,%d", x, y_value[x - 2]);

/*****
/*          DONE          */
/*****
puts("D,6,"); /* ALPHA: on, GRAPHICS: on, ERASE: off */
puts("E"); /* Exit Graphics Mode. */
)

```

```

/*****
/*
/*      NAME:      RIGHT.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 15
/*      FUNCTION:  Moves vertical CURSOR to right by amount given in
/*                  "input2". Movement past sample #500 causes a new 500 sample
/*                  graph to be displayed. CURSOR is placed at first sample of
/*                  graph.
/*      INPUTS:    NONE.
/*      OUTPUTS:   Draws new CURSOR on CRT screen. May result in a new
/*                  500 sample graph being displayed also.
/*      GLOBAL VARIABLES USED:  cursor, l_cursor, input1, input2,
/*                  x_axis, l_temp, max_samples.
/*      GLOBAL VARIABLES CHANGED:  cursor, l_cursor, x_axis, l_temp.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  plot().
/*      CALLING MODULES:  graphics().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  GLOBAL VARIABLES
/*
*****/
#include "speech.h"          /* Contains all GLOBAL VARIABLES.

```

```

/*****
/*
/*                  FUNCTION:  RIGHT()
/*
*****/
right()
{
    /*****
    /*                  LOCAL VARIABLES
    *****/
    char one[4];            /* Holds LONG (32 bit) version of 1.
    char two[4];            /* Holds LONG (32 bit) version of 2.
    char hundred[4];        /* Holds LONG (32 bit) version of 100.
    char five_hundred[4];   /* Holds LONG (32 bit) version of 500.
    char l_input2[4];       /* Holds LONG (32 bit) version of
                           /*   global variable "input2".
    char max[4];            /* Holds LONG (32 bit) version of
                           /*   maximum number of samples which
                           /*   can be stored.

```

```

/*****
/*          INITIALIZE VARIABLES          */
*****/
itol(one,1);          /* Convert 1 into a 32 bit integer. */
itol(two,2);          /* Convert 2 into a 32 bit integer. */
itol(hundred,100);    /* Convert 100 into a 32 bit integer. */
itol(five_hundred,500); /* Convert 500 into a 32 bit integer. */
itol(l_cursor,cursor); /* Convert cursor to a 32 bit integer. */
itol(l_input2,input2); /* Convert input2 to a 32 bit integer. */

ladd(l_temp,x_axis,l_cursor); /* Form sample number where cursor */
lsub(l_temp,l_temp,two);      /* is to be moved. */
ladd(l_temp,l_temp,l_input2);

atol(max,max_samples); /* Form 32 bit version of max number */
                        /* of samples which can be stored. */

/*****
/*          MOVE CURSOR TO THE RIGHT          */
*****/
if(lcomp(l_temp,max) == 1) /* If (l_temp > max), lcomp = 1 and */
;                          /* data sample can not exist. Return. */

else if(cursor < input1) { /* If room to move cursor to right and */
                          /* not go past edge of graph. */
    puts("\033l,");        /* Enter Graphics Mode. */
    puts("N,170,");        /* Primary Line Style: DASHED LINE. */
    puts("I,2,");          /* Line Type: COMPLEMENT. */

                          /* Remove existing "cursor line" and */
                          /* draw new "cursor line". */
    printf("P,%d,246,L,%d,47,",cursor,cursor);
    cursor = cursor + input2;
    printf("P,%d,246,L,%d,47,",cursor,cursor);
    puts("E");             /* Exit Graphics Mode. */

                          /* Update CURSOR value on CRT screen. */
    itol(l_cursor,cursor);
    ladd(l_temp,x_axis,l_cursor);
                          /* Line 23, Column 11. Update CURSOR. */
    puts("\033Y6*");
    printf("%-6s",ltoa(a_temp,lsub(l_temp,l_temp,two)));

} else {                  /* Moving cursor past right edge of */
                          /* graph, redraw graph. */
    cursor = 2;           /* Set cursor for 1st pixel of graph. */
                          /* Update CURSOR value on CRT screen. */

    itol(l_cursor,cursor);
    ladd(x_axis,x_axis,five_hundred);
    ladd(l_temp,x_axis,l_cursor);
                          /* Line 23, Column 11. Update CURSOR. */
    puts("\033Y6*");
    printf("%-6s",ltoa(a_temp,lsub(l_temp,l_temp,two)));
                          /* Line 22, Column 1. Update x-axis */
                          /* tick numbers. */

```

```

puts("\033Y5 ");
printf("      %-6s", ltoa(a_temp,x_axis));
lsub(l_temp,x_axis,one);
printf("%11s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%12s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%13s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%12s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%13s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));

plot();                                /* Plot next 500 samples on graph. */

puts("\033l,");                        /* Enter Graphics Mode. */
puts("N,170,");                        /* Primary Line Style: DASHED LINE. */
puts("I,2,");                          /* Line Type: COMPLEMENT. */
                                      /* Draw new "cursor line". */
printf("P,%d,246,L,%d,47,",cursor,cursor);
puts("E");                             /* Exit Graphics Mode. */
)
)

```

```

/*****
/*
/*      NAME:      LEFT.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 16
/*      FUNCTION:  Moves vertical CURSOR to left by amount given in
/*                  "input2". Movement past first sample causes a new 500 sample
/*                  graph to be displayed. CURSOR is placed at last sample of new
/*                  graph.
/*      INPUTS:    NONE.
/*      OUTPUTS:   Draws new CURSOR on CRT screen. May also result in a
/*                  new 500 sample graph being displayed.
/*      GLOBAL VARIABLES USED:  cursor, l_cursor, input1, input2,
/*                               x_axis, l_temp, max_samples.
/*      GLOBAL VARIABLES CHANGED:  cursor, l_cursor, x_axis, l_temp.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  plot().
/*      CALLING MODULES:  graphics().
/*
/*      AUTHOR:  CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:  AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  GLOBAL VARIABLES
/*
*****/
#include "speech.h"          /* Contains all GLOBAL VARIABLES.

```

```

/*****
/*
/*                  FUNCTION:  LEFT()
/*
*****/
left()
{
    /*****
    /*                  LOCAL VARIABLES
    /*
    *****/
    char one[4];             /* Holds LONG (32 bit) version of 1.
    char two[4];             /* Holds LONG (32 bit) version of 2.
    char hundred[4];        /* Holds LONG (32 bit) version of 100.
    char five_hundred[4];   /* Holds LONG (32 bit) version of 500.
    char l_input2[4];        /* Holds LONG (32 bit) version of
                               /* global variable "input2".

```

```

/*****
/*          INITIALIZE VARIABLES          */
*****/
itol(one,1);          /* Convert 1 into a 32 bit integer. */
itol(two,2);          /* Convert 2 into a 32 bit integer. */
itol(hundred,100);    /* Convert 100 into a 32 bit integer. */
itol(five_hundred,500); /* Convert 500 into a 32 bit integer. */
itol(l_cursor,cursor); /* Convert cursor into a 32 bit integer.*/
itol(l_input2,input2); /* Convert input2 into a 32 bit integer.*/

ladd(l_temp,x_axis,l_cursor); /* Form sample number where cursor */
lsub(l_temp,l_temp,two);      /* is to be moved. */
lsub(l_temp,l_temp,l_input2);

/*****
/*          MOVE CURSOR TO THE LEFT          */
*****/
if(lcomp(l_temp,one) == -1) /* If (l_temp < one), lcomp = -1 and */
; /* data sample can not exist. Return. */

else if(cursor > input1) { /* If room to move cursor to left and */
/* not go past edge of graph. */
puts("\033I,"); /* Enter Graphics Mode. */
puts("N,170,"); /* Primary Line Style: DASHED LINE. */
puts("I,2,"); /* Line Type: COMPLEMENT. */

/* Remove existing "cursor line" and */
/* draw new "cursor line". */
printf("P,%d,246,L,%d,47,",cursor,cursor);
cursor = cursor - input2;
printf("P,%d,246,L,%d,47,",cursor,cursor);
puts("E"); /* Exit Graphics Mode. */
/* Update CURSOR value on CRT screen. */
itol(l_cursor,cursor);
ladd(l_temp,x_axis,l_cursor);
/* Line 23, Column 11. Update CURSOR. */
puts("\033Y6*");
printf("%-6s",ltoa(a_temp,lsub(l_temp,l_temp,two)));

} else { /* Moving cursor past left edge of */
/* graph, redraw graph. */
cursor = 501; /* Set cursor for last pixel of graph. */
/* Update CURSOR value on CRT screen. */
itol(l_cursor,cursor);
lsub(x_axis,x_axis,five_hundred);
ladd(l_temp,x_axis,l_cursor);
/* Line 23, Column 11. Update CURSOR. */
puts("\033Y6*");
printf("%-6s",ltoa(a_temp,lsub(l_temp,l_temp,two)));
/* Line 22, Column 1. Update x-axis */
/* tick numbers. */
puts("\033Y5 ");
printf("%-6s", ltoa(a_temp,x_axis));
lsub(l_temp,x_axis,one);

```

```

printf("%11s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%12s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%13s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%12s", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));
printf("%13s      ", ltoa(a_temp,ladd(l_temp,l_temp,hundred)));

plot();                      /* Plot next 500 samples on graph.      */

puts("\033l,");              /* Enter Graphics Mode.          */
puts("N,170,");              /* Primary Line Style: DASHED LINE. */
puts("I,2,");                /* Line Type: COMPLEMENT.        */
                             /* Draw new "cursor line".       */
printf("P,%d,246,L,%d,47,",cursor,cursor);
puts("E");                   /* Exit Graphics Mode.           */

```

```

/*****
/*
/*      NAME:      VOLTLINE.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 17
/*      FUNCTION:  Moves horizontal Volt Line up or down by amount
/*                  given in "input1". Volt Line is not allowed to move off the
/*                  graph.
/*      INPUTS:    NONE.
/*      OUTPUTS:   Draws new Volt Line on CRT screen.
/*      GLOBAL VARIABLES USED:  voltline, input1.
/*      GLOBAL VARIABLES CHANGED:  voltline.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  NONE.
/*      CALLING MODULES:  graphics().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  GLOBAL VARIABLES
/*
*****/
#include "speech.h"          /* Contains all GLOBAL VARIABLES.

```

```

/*****
/*
/*                  FUNCTION:  VOLTLINE()
/*
*****/
voltline()
{
    /*****
    /*
    /*                  MOVE VOLTAGE LINE
    *****/
    /* Volt Line must not move off graph.
    if((volt_line + input1 >= 47) && (volt_line + input1 <= 246)) {
        puts("\033l,");          /* Enter Graphics Mode.
        puts("N,170,");          /* Primary Line Style: DASHED LINE.
        puts("I,2,");            /* Line Type: COMPLEMENT.
        /* Remove existing "voltage line" and
        /* draw new "voltage line".
        printf("P,6,%d,L,501,%d,",volt_line,volt_line);
        volt_line = volt_line + input1;
        printf("P,6,%d,L,501,%d,",volt_line,volt_line);
        puts("E");              /* Exit Graphics Mode.
    }
}

```

```

/*****
/*
/*      NAME:      INPUT.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 18
/*      FUNCTION:  Allows user to: (1) Select "analog input channel" to
/*                  be sampled, (2) Select data "sampling rate", (3) Select total
/*                  "number of samples" to be taken, and (4) Select "first and
/*                  last sample" to be converted during D/A conversion.
/*      INPUTS:    User entered alphanumerics from H-19 keyboard.
/*      OUTPUTS:   User prompts displayed on CRT screen.
/*      GLOBAL VARIABLES USED:  is_analog, channel, chan_num, rate,
/*                  rate_num, max_rate, is_graphics, is_digital, begin_at, start,
/*                  max_samples, samples, finish.
/*      GLOBAL VARIABLES CHANGED: channel, chan_num, rate, rate_num,
/*                  begin_at, start, samples, finish.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  NONE.
/*      CALLING MODULES:  analog(), digital(), graphics().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station.  Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
*****/

```

```

#define ESCAPE      27 /* H-19 CRT ASCII "escape" code.
#define R_VIDEO     0x70 /* H-19 CRT code - enter reverse video mode.
#define N_VIDEO     0x71 /* H-19 CRT code - enter normal video mode.
#define TRUE        1 /* Logic "true" is a 1.
#define BACKSPACE   0x08 /* ASCII hex code for "Backspace" key.
#define DELETE      0x7F /* ASCII hex code for "Delete" key.

```

```

/*****
/*
/*                  GLOBAL VARIABLES
/*
*****/

```

```

#include "speech.h" /* Contains all GLOBAL VARIABLES.

```

```

/*****
/*
/*          FUNCTION:  INPUT()
/*
/*****
input()
{
    /*****
    /*          LOCAL VARIABLES
    /*****
    int i;                /* Indexing variable used in "for" loop.*/
    char c;               /* Holds keyboard response to getchar().*/
    char max[4];          /* Holds LONG (32 bit) version of max   */
                        /* number of samples that can be stored*/
                        /* in extended memory.                */
    int compare;          /* Used to determine if an "input" is   */
                        /* larger than maximum value allowed. */

    putchar(ESCAPE);      /* Display user prompt in reverse video.*/
    putchar(R_VIDEO);
    puts("\n\nPress <CR> for default value.\n");
    putchar(ESCAPE);
    putchar(N_VIDEO);

    /*****
    /*          SELECT ANALOG CHANNEL
    /*****
                        /* When requested, display prompt for   */
                        /* ANALOG CHANNEL.
                        */

    if(is_analog == TRUE) {
E1:    printf("\nANALOG CHANNEL to be sampled [%s", channel);
        printf("]:  .. <CR>\b\b\b\b\b\b\b\b\b\b");
                        /* Skip all keyboard entries until first*/
                        /* "digit" or <CR> arrives.
                        */
        i = 0;
        while(((c=getchar()) < '0' || c > '9') && (c != '\n')) {
            if(c == BACKSPACE || c == DELETE)
                puts("");
            else
                puts("\b.\b");
        }

                        /* Enter ANALOG CHANNEL to be sampled   */
                        /* Skip all keyboard entries except     */
                        /* "digits" and <CR>.
                        */

        if(c != '\n') {
            channel[i] = c;
            for(i = 1; (c = getchar()) != '\n' && i < 2; ++i) {
                if(c == BACKSPACE || c == DELETE)
                    --i;
                else if(c < '0' || c > '9') {
                    puts("\b.\b");
                    --i;
                } else
                    channel[i] = c;
            }
        }
    }
}

```

```

        channel[i] = '\0';
    }
    chan_num = atoi(channel); /* Convert ASCII "channel" to 16 bit    */
                               /* integer value.                    */
                               /* Check for and respond to any errors. */
    if(chan_num > 15) {
        puts("\nERROR: MAGIC only allows channels 0 thru 15 inclusive.\n");
        goto E1;
    }
}

/*****
/*                                SELECT SAMPLING RATE                                */
*****/

/* When requested, display appropriate */
/* user prompt for SAMPLING RATE.      */

E2:if(is_graphics ==TRUE) {
    printf("\nWelcome to GRAPHICS, %s. \n", name);
    printf("\nDuring any Analog Output, the SAMPLING RATE ");
    printf("should be [%s", rate);
} else
    printf("\nSAMPLING RATE in samples/sec [%s", rate);

printf("]: ..... <CR>\b\b\b\b\b\b\b\b\b\b\b\b");
/* Skip all keyboard entries until first*/
/* "digit" or <CR> arrives.              */

i = 0;
while(((c=getchar()) < '0' || c > '9') && (c != '\n')) {
    if(c == BACKSPACE || c == DELETE)
        puts("");
    else
        puts("\b.\b");
}

/* Enter SAMPLING RATE.                */
/* Skip all keyboard entries except    */
/* "digits" and <CR>.                  */

if(c != '\n') {
    rate[i] = c;
    for(i = 1; (c = getchar()) != '\n' && i < 5; ++i) {
        if(c == BACKSPACE || c == DELETE)
            --i;
        else if(c < '0' || c > '9') {
            puts("\b.\b");
            --i;
        } else
            rate[i] = c;
    }
    rate[i] = '\0';
}

```

```

atol(rate_num, rate);          /* Convert ASCII "Sampling Rate" to      */
                                /* 32 bit integer value.          */
atol(max, max_rate);           /* Convert ASCII "Max Sampling Rate" to */
                                /* 32 bit integer value.          */
                                /* Check for and respond to any error. */
                                /* NOTE: compare = 1 if (rate_num > max) */
compare = lcomp(rate_num, max);
if(compare == TRUE) {
    puts("\nERROR: Maximum sampling rate for MAGIC is 31,700.\n");
    goto E2;
}

/*****
/*          SELECT FIRST DIGITAL-TO-ANALOG SAMPLE          */
*****/

                                /* When requested, display prompt for */
                                /* "first sample" to be D/A processed. */
E3:if((is_digital == TRUE) || (is_graphics == TRUE)) {
    if(is_graphics == TRUE) {
        puts("\n500 samples at a time are displayed.\n");
        puts("\nMAGIC should START with sample ");
        printf("number [%s", begin_at);
    } else
        printf("\nSTART with sample number [%s", begin_at);

    printf("]: ..... <CR>\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b");
                                /* Skip all keyboard entries until first */
                                /* "digit" or <CR> arrives.          */

    i = 0;
    while(((c=getchar()) < '0' || c > '9') && (c != '\n')) {
        if(c == BACKSPACE || c == DELETE)
            puts("");
        else
            puts("\b.\b");
    }

                                /* Enter "FIRST SAMPLE" to be processed. */
                                /* Skip all keyboard entries except      */
                                /* "digits" and <CR>.          */

    if(c != '\n') {
        begin_at[i] = c;
        for(i = 1; (c = getchar()) != '\n' && i < 6; ++i) {
            if(c == BACKSPACE || c == DELETE)
                --i;
            else if(c < '0' || c > '9') {
                puts("\b.\b");
                --i;
            } else
                begin_at[i] = c;
        }
        begin_at[i] = '\0';
    }
}

```

```

        atol(start, begin_at);      /* Convert ASCII "first sample" number */
        atol(max, max_samples);     /* to 32 bit integer value. */
        /* Convert ASCII "maximum sample number" */
        /* to 32 bit integer value. */
        /* Check for and respond to any error. */
        /* NOTE: compare = 1 if (start > max). */
        compare = lcomp(start, max);
        if(compare == TRUE) {
            puts("\nERROR: MAGIC limits START sample to under 163,840.\n");
            goto E3;
        }
    }

    if(is_graphics == TRUE)          /* Skip rest of function because "last */
        goto BYE;                   /* sample" to be processed is provided */
                                    /* by graphics(). */

    /***** SELECT NUMBER OF SAMPLES TO PROCESS *****/
    /* When requested, display user prompt */
    /* for "last sample" to be processed. */

E4:if(is_analog == TRUE)
    printf("\nNUMBER OF SAMPLES to be taken [%s", samples);
else if(is_digital == TRUE)
    printf("\nFINISH with sample number [%s", samples);

printf("]: ..... <CR>\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b\b");
/* Skip all keyboard entries until first */
/* "digit" or <CR> arrives. */

i = 0;
while(((c=getchar()) < '0' || c > '9') && (c != '\n')) {
    if(c == BACKSPACE || c == DELETE)
        puts("");
    else
        puts("\b.\b");
}

/* Enter "LAST SAMPLE" to be processed. */
/* Skip all keyboard entries except */
/* "digits" and <CR>. */

if(c != '\n') {
    samples[i] = c;
    for(i = 1; (c = getchar()) != '\n' && i < 6; ++i) {
        if(c == BACKSPACE || c == DELETE)
            --i;
        else if(c < '0' || c > '9') {
            puts("\b.\b");
            --i;
        } else
            samples[i] = c;
    }
    samples[i] = '\0';
}

```

```

atol(finish, samples);      /* Convert ASCII "last sample" to      */
                             /* 32 bit integer value.      */
atol(max, max_samples);     /* Convert ASCII "maximum sample number" */
                             /* to 32 bit integer value.      */
                             /* Check for and respond to any error. */
                             /* NOTE: compare = 1 if (finish > max). */

compare = lcomp(finish, max);
if(compare == TRUE && is_analog == TRUE) {
    puts("\nERROR: MAGIC limits the number of samples to 163,840.\n");
    goto E4;
} else if(compare == TRUE && is_digital == TRUE) {
    puts("\nERROR: MAGIC limits the last sample to 163,840.\n");
    goto E4;
}

/*****
/*                               DONE                               */
*****/

BYE: ;
}

```

```

/*****
/*
/*      NAME:      DMA.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 19
/*      FUNCTION:  Initializes Direct Memory Access (DMA) chip's
/*                  internal register's to: (1) Support analog-to-digital sampling
/*                  (2) Support digital-to-analog conversion, (3) Provide 2048
/*                  byte memory-to-memory data transfer to support store() and
/*                  retrieve(), and (4) Provide 64K byte memory-to-memory data
/*                  transfer to clear extended memory boards.
/*      INPUTS:    NONE.
/*      OUTPUTS:   NONE.
/*      GLOBAL VARIABLES USED:  is_mem_mem, is_graphics, is_analog,
/*                  is_digital, is_clearmem.
/*      GLOBAL VARIABLES CHANGED:  NONE.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  nop().
/*      CALLING MODULES:  analog(), digital(), store(), retrieve(),
/*                  plot(), clearmem().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station.  Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
*****/

#define  RESET      0x9D  /* DMA Master Reset.
/*
#define  COMMAND    0x98  /* Write COMMAND REGISTER.
/*
#define  A_OR_D     0x80  /* Set Command Reg for A/D or D/A operation.
/*
#define  MEM_MEM    0x81  /* Set Command Reg for MEM-MEM transfer.
/*
#define  MEM_CLEAR  0x83  /* Set Command Reg to NULL Extended Memory.
/*

#define  MASK_FOR   0x9F  /* Write all MASK REGISTERS.
/*
#define  ANALOG     0x0B  /* Set DMA Mask for A/D operation.
/*
#define  DIGITAL    0x07  /* Set DMA Mask for D/A operation.
/*
#define  MEM_XFER   0x0F  /* Set DMA Mask for MEM-MEM transfer.
/*

#define  MODE       0x9B  /* Write MODE REGISTER.
/*

#define  CHO_MREG   0x88  /* Set Chan 0 MODE REG for mem-mem operation.
/*
#define  CHO_ADDR   0x90  /* Channel 0 ADDRESS REGISTER.
/*
#define  CHO_BYTE   0x91  /* Channel 0 BYTE COUNT REGISTER.
/*

```

```

#define CH1_MREG    0x85  /* Set Chan 1 MODE REG for mem-mem operation. */
#define CH1_ADDR    0x92  /* Channel 1 ADDRESS REGISTER. */
#define CH1_BYTE    0x93  /* Channel 1 BYTE COUNT REGISTER. */

#define CH2_MREG    0x56  /* Set Chan 2 MODE REG for A/D operation. */
#define CH2_ADDR    0x94  /* Channel 2 ADDRESS REGISTER. */
#define CH2_BYTE    0x95  /* Channel 2 BYTE COUNT REGISTER. */

#define CH3_MREG    0x5B  /* Set Chan 3 MODE REG for D/A operation. */
#define CH3_ADDR    0x96  /* Channel 3 ADDRESS REGISTER. */
#define CH3_BYTE    0x97  /* Channel 3 BYTE COUNT REGISTER. */

#define TRUE        1     /* Logic "true" is a 1. */

#define ZERO        0x00  /* Used to initialize memory location. */
#define BYTE_COUNT  0xFF  /* Used to initialize byte count. */

```

```

/*****
/*                                GLOBAL VARIABLES                                */
*****/
#include "speech.h"                /* Contains all GLOBAL VARIABLES. */

```

```

/*****
/*                                FUNCTION: DMA()                                */
*****/

/* NOTE: CHANNEL 0: MEMORY-MEMORY TRANSFER (SOURCE). */
/* CHANNEL 1: MEMORY-MEMORY TRANSFER (DESTINATION). */
/* CHANNEL 2: ANALOG-TO-DIGITAL CONVERSION. */
/* CHANNEL 3: DIGITAL-TO-ANALOG CONVERSION. */
/*****

```

```

dma()
{
    /*****
    /*                                LOCAL VARIABLES                                */
    *****/
    char xfer_lo;                /* Holds number of bytes to transfer */
    char xfer_hi;                /* during Memory-to-Memory operations. */

    /*****
    /*                                LOAD DMA COMMAND REGISTER                                */
    *****/
    outp(RESET);                /* RESET DMA chip. */
    nop();                      /* NOTE: A small delay is required */
                                /* between loading of DMA's internal */
                                /* registers. */
}

```

```

if((is_mem_mem == TRUE) || (is_graphics == TRUE)) {
    outp(COMMAND, MEM_MEM);
    nop();
} else if((is_analog == TRUE) || (is_digital == TRUE)) {
    outp(COMMAND, A_OR_D);
    nop();
} else {
    outp(COMMAND, MEM_CLEAR);
    nop();
}

/*****
/*          LOAD DMA FOR: MEMORY-TO-MEMORY TRANSFER          */
*****/

/* Determine the number of bytes to be */
/* moved during a Memory-to-Memory    */
/* transfer.                            */

if(is_mem_mem == TRUE) {
    xfer_lo = 0x00; /* (0800)H = 2048, number bytes moved */
    xfer_hi = 0x08; /* during STORE() and RETRIEVE(). */
} else if(is_graphics == TRUE) {
    xfer_lo = 0xE8; /* (03E8)H = 1000, number of bytes to */
    xfer_hi = 0x03; /* be moved during PLOT(). */
} else {
    xfer_lo = 0xFF; /* (FFFF)H = 64K, number of bytes to be */
    xfer_hi = 0xFF; /* NULLED during CLEARMEM(). */
}

if((is_mem_mem == TRUE) || (is_graphics == TRUE) || (is_clearmem)) {
    outp(MODE, CHO_MREG); /* Load Group 0 MODE REGISTER. */
    nop(); /* (SOURCE.) */
    outp(CHO_ADDR, from[3]); /* Load "Address Register" with */
    nop(); /* with address of where first */
    outp(CHO_ADDR, from[2]); /* byte is to "come from". */
    nop();
    outp(CHO_BYTE, xfer_lo); /* Load "Word Count Register" */
    nop(); /* with number of bytes to be */
    outp(CHO_BYTE, xfer_hi); /* transferred. */

    outp(MODE, CH1_MREG); /* Load Group 1 MODE REGISTER. */
    nop(); /* (DESTINATION.) */
    outp(CH1_ADDR, to[3]); /* Load "Address Register" with */
    nop(); /* address where 1st byte is */
    outp(CH1_ADDR, to[2]); /* "moved to". */
    nop();
    outp(CH1_BYTE, xfer_lo); /* Load "Word Count Register" */
    nop(); /* with number of bytes to be */
    outp(CH1_BYTE, xfer_hi); /* transferred. */
    nop();
}

```

```

/*****
/*          LOAD DMA FOR: ANALOG-TO-DIGITAL SAMPLING          */
*****/
if(is_analog == TRUE) {
    outp(MODE, CH2_MREG);          /* Load Group 2 MODE REGISTER. */
    nop();
    outp(CH2_ADDR, ZERO);          /* Load "Address Register" with */
    nop();                        /* address of 1st byte (0000H) */
    outp(CH2_ADDR, ZERO);          /* on extended memory boards. */
    nop();
    outp(CH2_BYTE, BYTE_COUNT);    /* Load "Word Count Register" */
    nop();                        /* with number of bytes (64K) */
    outp(CH2_BYTE, BYTE_COUNT);    /* on extended memory boards. */
    nop();
}

/*****
/*          LOAD DMA FOR: DIGITAL-TO-ANALOG CONVERSION          */
*****/
if(is_digital == TRUE) {
    outp(MODE, CH3_MREG);          /* Load Group 3 MODE REGISTER. */
    nop();
    outp(CH3_ADDR, ZERO);          /* Load "Address Register" with */
    nop();                        /* address of 1st byte (0000H) */
    outp(CH3_ADDR, ZERO);          /* on extended memory boards. */
    nop();
    outp(CH3_BYTE, BYTE_COUNT);    /* Load "Word Count Register" */
    nop();                        /* with number of bytes (64K) */
    outp(CH3_BYTE, BYTE_COUNT);    /* on extended memory boards. */
    nop();
}

/*****
/*          MASK DMA REGISTERS          */
*****/
if(is_analog == TRUE) {
    outp(MASK_FOR, ANALOG);        /* Allow only analog-to-digital */
    nop();                        /* requests. */
} else if (is_digital == TRUE) {
    outp(MASK_FOR, DIGITAL);       /* Allow only digital-to-analog */
    nop();                        /* requests. */
} else {
    outp(MASK_FOR, MEM_XFER);      /* Allow only software memory- */
    nop();                        /* to-memory transfer requests.*/
}
)

```

```

/*****
/*
/*      NAME:      TIMING.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 20
/*      FUNCTION:  Initializes System Timing Controller (STC) chip's
/*                  internal registers to: (1) Provide A/D or D/A clock pulses at
/*                  user specified triggering rate, (2) Disable A/D or D/A
/*                  capability after user specified number of data samples are
/*                  processed, and (3) Provide by-passing of unrequired data
/*                  samples at the beginning of a memory board during digital-to-
/*                  analog output operation.
/*      INPUTS:  NONE.
/*      OUTPUTS: NONE.
/*      GLOBAL VARIABLES USED:  limit1A, limit2A, limit3A, limit4A,
/*                  limit5A, limit6A, limit7A, limit8A, is_analog, finish,
/*                  is_digital, start, crystal, rate_num.
/*      GLOBAL VARIABLES CHANGED:  NONE.
/*      FILES READ:  NONE.
/*      FILES WRITTEN:  NONE.
/*      MODULES CALLED:  NONE.
/*      CALLING MODULES:  analog(), digital().
/*
/*      AUTHOR:  CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:  AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station.  Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
*****/

```

```

#define  STC_D      0x10  /* Address System Timing Controller (STC)
/* chip select - data register transfer.
#define  STC_C      0x11  /* Address STC chip select - control
/* register transfer.
#define  RESET      0xFF  /* STC Master Reset code.

#define  MM_REG      0x17  /* MASTER MODE register's address.
#define  MM_LOBYTE   0x0C  /* Reg 1 & 2: Set to compare separately.
#define  MM_HIBYTE   0xC1

#define  REG1_CM      0x01  /* Address COUNTER MODE register, group 1.
#define  CM1_LOBYTE   0x29  /* Set for active HI pulse out on a true
#define  CM1_HIBYTE   0x00  /* compare.
#define  REG1_COMPARE 0x07  /* Address COMPARE register, group 1.
#define  REG1_LOAD    0x09  /* Address LOAD register, group 1.

```

```

#define REG2_CM      0x02 /* Address COUNTER MODE register, group 2. */
#define CM2_LOBYTE   0x29 /* Set for active HI pulse out on a true */
#define CM2_HIBYTE   0x00 /* compare. */
#define REG2_COMPARE 0x0F /* Address COMPARE register, group 2. */
#define REG2_LOAD    0x0A /* Address LOAD register, group 2. */

#define REG4_CM      0x04 /* Address COUNTER MODE register, group 4. */
#define CM4_LOBYTE   0x02 /* Set for active HI toggle. */
#define CM4_HIBYTE   0x14
#define REG4_LOAD    0x0C /* Address LOAD register, group 4. */

#define REG5_CM      0x05 /* Address COUNTER MODE register, group 5. */
#define CM5_LOBYTE   0x21 /* Set for active HI pulse out. */
#define CM5_HIBYTE   0x0B
#define REG5_LOAD    0x0D /* Address LOAD register, group 5. */

#define TRUE         1 /* Logic "true" is a 1. */
#define ZERO         0x00 /* Makes program easier to read. */
#define LOAD         0x5B /* Load STC registers 1,2,4,5. */
#define ARM          0x2B /* Arm STC registers 1,2,4. */

```

```

/*****
/*                               GLOBAL VARIABLES                               */
*****/
#include "speech.h" /* Contains all GLOBAL VARIABLES. */

```

```

/*****
/*                               FUNCTION: TIMING()                               */
*****/

```

```

/*****
/* NOTE:  REGISTERS 1&2 -- FINISH COUNT                                     */
/*          REGISTER 4   -- D/A OFFSET                                     */
/*          REGISTER 5   -- CLOCK                                         */
*****/

```

timing()

```

{
/*****
/*                               LOCAL VARIABLES                               */
*****/
char limit1[4]; /* Holds 32 bit version of global "limit1A[7]". */
char limit2[4]; /* Holds 32 bit version of global "limit2A[7]". */
char limit3[4]; /* Holds 32 bit version of global "limit3A[7]". */
char limit4[4]; /* Holds 32 bit version of global "limit4A[7]". */
char limit5[4]; /* Holds 32 bit version of global "limit5A[7]". */
char limit6[4]; /* Holds 32 bit version of global "limit6A[7]". */
char limit7[4]; /* Holds 32 bit version of global "limit7A[7]". */
char limit8[4]; /* Holds 32 bit version of global "limit8A[7]". */

char one[4]; /* Holds LONG (32 bit) version of 1. */
char two[4]; /* Holds LONG (32 bit) version of 2. */

```

```

char stop_at[4];      /* Holds 32 bit version of number of "data    */
                      /* samples" to be taken plus one.                */
char start_at[4];     /* Holds 32 bit version of number of "data    */
                      /* samples" to be by-passed on a memory board  */
                      /* before a digital-to-analog output begins.    */
char frequency[4];    /* Holds 32 bit version of global "crystal".    */
char clock[4];        /* Holds 32 bit version of STC "crystal" pulses */
                      /* which must occur before a TRIGGER clock     */
                      /* pulse is sent out by STC Register 5.        */

int is1;              /* is1 thru is8 are used to determine on which */
int is2;              /* memory board a digital-to-analog conversion */
int is3;              /* is to begin. is1 and is2 determine if the   */
int is4;              /* first sample to be outputted is on extended */
int is5;              /* memory board #1. is3 and is4 determine if   */
int is6;              /* the first sample to be outputted is on     */
int is7;              /* extended memory board #2. And so on.        */
int is8;

```

```

/*****
/*          INITIALIZE VARIABLES          */
*****/

```

```

atol(limit1,limit1A); /* Form 32 bit version of "limit1A".    */
atol(limit2,limit2A); /* Form 32 bit version of "limit2A".    */
atol(limit3,limit3A); /* Form 32 bit version of "limit3A".    */
atol(limit4,limit4A); /* Form 32 bit version of "limit4A".    */
atol(limit5,limit5A); /* Form 32 bit version of "limit5A".    */
atol(limit6,limit6A); /* Form 32 bit version of "limit6A".    */
atol(limit7,limit7A); /* Form 32 bit version of "limit7A".    */
atol(limit8,limit8A); /* Form 32 bit version of "limit8A".    */

itol(one,1);          /* Convert 1 into a 32 bit integer.      */
itol(two,2);          /* Convert 2 into a 32 bit integer.      */

```

```

/*****
/*          DETERMINE NUMBER OF SAMPLES TO CLOCK IN OR OUT          */
*****/

```

```

if(is_analog == TRUE) /* For A/D, clock one more time than    */
    ladd(stop_at,finish,one); /* total number of samples requested.  */

else if(is_digital == TRUE) { /* For D/A, first determine which    */
    /* memory board the 1st sample is on. */
    is1 = lcomp(start,limit1); /* For (start > limit1), lcomp = 1.    */
    is2 = lcomp(start,limit2); /* For (start = limit1), lcomp = 0.    */
    is3 = lcomp(start,limit3); /* For (start < limit1), lcomp = -1.    */
    is4 = lcomp(start,limit4);
    is5 = lcomp(start,limit5);
    is6 = lcomp(start,limit6);
    is7 = lcomp(start,limit7);
    is8 = lcomp(start,limit8);
}

```

```

/* Here is where the determination is */
/* really made. Total samples include */
/* those by-passed at the beginning of */
/* a memory board. */
if((isl == 1 || isl == 0) && (is2 == -1 || is2 == 0)) {
    ladd(stop_at,finish,one);
} else if((is3 == 1 || is3 == 0) && (is4 == -1 || is4 == 0)) {
    lsub(stop_at,finish,limit2);
    ladd(stop_at,stop_at,one);
} else if((is5 == 1 || is5 == 0) && (is6 == -1 || is6 == 0)) {
    lsub(stop_at,finish,limit4);
    ladd(stop_at,stop_at,one);
} else if((is7 == 1 || is7 == 0) && (is8 == -1 || is8 == 0)) {
    lsub(stop_at,finish,limit6);
    ladd(stop_at,stop_at,one);
} else {
    lsub(stop_at,finish,limit8);
    ladd(stop_at,stop_at,one);
}
}

/*****
/*                                LOAD STC REGISTERS                                */
*****/

outp(STC_C, RESET);          /* Master Reset the STC. */
outp(STC_C, MM_REG);         /* Load the MASTER MODE register. */
outp(STC_D, MM_LOBYTE);
outp(STC_D, MM_HIBYTE);

outp(STC_C, REG1_CM);        /* Load the COUNTER MODE register for */
outp(STC_D, CM1_LOBYTE);    /* register group number 1. */
outp(STC_D, CM1_HIBYTE);

outp(STC_C, REG1_COMPARE);   /* Load the COMPARE register for */
outp(STC_D, stop_at[3]);    /* register group number 1. */
outp(STC_D, stop_at[2]);

outp(STC_C, REG1_LOAD);      /* Load the LOAD register for register */
outp(STC_D, ZERO);          /* group number 1. */
outp(STC_D, ZERO);

outp(STC_C, REG2_CM);        /* Load the COUNTER MODE register for */
outp(STC_D, CM2_LOBYTE);    /* register group number 2. */
outp(STC_D, CM2_HIBYTE);

outp(STC_C, REG2_COMPARE);   /* Load the COMPARE register for */
outp(STC_D, stop_at[1]);    /* register group number 2. */
outp(STC_D, stop_at[0]);

outp(STC_C, REG2_LOAD);      /* Load the LOAD register for register */
outp(STC_D, ZERO);          /* group number 2. */
outp(STC_D, ZERO);

```

```

outp(STC_C, REG4_CM);          /* Load the COUNTER MODE register for */
outp(STC_D, CM4_LOBYTE);      /* register group number 4.          */
outp(STC_D, CM4_HIBYTE);

lmul(start_at,start,two);      /* Determine the samples to be by-passed*/
lsub(start_at,start_at,one);   /* before a D/A conversion begins.    */
outp(STC_C, REG4_LOAD);        /* Load the LOAD register for register */
outp(STC_D, start_at[3]);      /* group number 4.                    */
outp(STC_D, start_at[2]);

outp(STC_C, REG5_CM);          /* Load the COUNTER MODE register for */
outp(STC_D, CM5_LOBYTE);      /* register group number 5.          */
outp(STC_D, CM5_HIBYTE);

atoj(frequency,crystal);       /* Determine how often the trigger */
ldiv(clock,frequency,rate_num); /* clock pulse is to occur.        */
outp(STC_C, REG5_LOAD);        /* Load the LOAD register for register */
outp(STC_D, clock[3]);         /* group number 5.                  */
outp(STC_D, clock[2]);

outp(STC_C, LOAD);             /* Load STC register groups 1,2,4,5. */
outp(STC_C, ARM);             /* Arm STC register groups 1,2,4.    */
)

```

```

/*****
/*
/*      NAME:      CLEARMEM.C
/*      VERSION:   1.0
/*      DATE:      2 December 1983
/*      MODULE NUMBER: 21
/*      FUNCTION:  Places binary zero (00000000B) in each memory byte
/*                  of all extended memory boards.
/*      INPUTS: NONE.
/*      OUTPUTS: NONE.
/*      GLOBAL VARIABLES USED:  is_clearmem, from, to, mem_buffer.
/*      GLOBAL VARIABLES CHANGED: is_clearmem, from, to, mem_buffer.
/*      FILES READ:  NONE.
/*      FILES WRITTEN: NONE.
/*      MODULES CALLED: dma().
/*      CALLING MODULES: analog(), retrieve().
/*
/*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
/*
/*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated
/*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
/*
*****/

```

```

/*****
/*
/*                  SYMBOLIC CONSTANTS
/*
*****/

#define TRUE      1 /* Logic "true" is a 1.
#define FALSE     0 /* Logic "false" is a 0.
#define TOGGLE    0 /* TOGGLE may be any value. TOGGLE is used in
/*                  the 'value' position of the "C" statement:
/*                  outp(port,value).
#define HI_LO     0xB0 /* Address which toggles memory-to-memory data
/*                  transfer between "HI-memory to LO-memory"
/*                  mode and "LO-memory to HI-memory" mode.
#define CPU_MEM   0xD0 /* Address which toggles between "CPU or DMA"
/*                  actions and "MEMORY-TO-MEMORY" transfers.
#define EXTENDED  0x70 /* Address used to load Extended Address.
#define REQUEST   0x99 /* Address of software DREQ Request Register.
#define MEM_XFER  0x04 /* Software DMA CHAN-0 request, i.e. start
/*                  memory-to-memory transfer.

```

```

/*****
/*
/*                  GLOBAL VARIABLES
/*
*****/

#include "speech.h" /* Contains all GLOBAL VARIABLES.

```

```

/*****
/*
/*          FUNCTION: CLEARMEM()
/*
/*
/*****
clearmem()
(
    /*****
    /*          LOCAL VARIABLES
    /*****
    int i;          /* Index variable used in "for" loop.
    int holder;     /* Holds address found by *ptr_num.
                    /* Binary operations can be done on a
                    /* integer value, but not a pointer.
    char *ptr_num;  /* Used to find first address of
                    /* mem_buffer array.

    /*****
    /*          INITIALIZE VARIABLES
    /*****

    is_clearmem = TRUE;          /* Enable selected portion of other
                                /* functions used by clearmem().
    ptr_num = &mem_buffer[0];    /* Find memory address of buffer array.
    holder = ptr_num;

    from[3] = ptr_num;          /* Memory address where data is found.
    from[2] = holder >> 8;      /* Used by dma().
    from[1] = 0x00;
    from[0] = 0x00;

    mem_buffer[0] = 0x00;       /* Load data byte (00000000B) to be
                                /* transfered.

    to[3] = 0x00;               /* Initial memory address where data is
    to[2] = 0x00;               /* to be moved. Used by dma().
    to[1] = 0x01;
    to[0] = 0x00;

    /*****
    /*          CLEAR EXTENDED MEMORY
    /*****

    outp(CPU_MEM,TOGGLE);       /* Set hardware for "Memory-to-Memory"
                                /* data transfer.
    outp(HI_LO,TOGGLE);         /* Set hardware for "LO-memory to HI-
                                /* memory" data transfer.
    for(i = 1; i <= 5; ++i) {   /* Clear all extended memory boards.
        outp(EXTENDED,to[1]);   /* Load extended address of memory
                                /* board to be cleared.
        dma();                  /* Set DMA chip for data transfer.
        outp(REQUEST,MEM_XFER); /* Begin 64K Memory-to-Memory transfer.
        to[1] = to[1] + 0x01;    /* Form extended address of next memory
                                /* board to be cleared.
    }

```

```

/*****
/*                                     DONE                                     */
/*****
outp(HI_LO,TOGGLE);                /* Set hardware for "HI-memory to LO- */
/* memory" data transfer.          */
outp(CPU_MEM,TOGGLE);              /* Set hardware for "CPU or DMA"     */
/* operation.                      */
is_clearmem = FALSE;               /* Disable selected portions of other */
/* functions used by clearmem().    */
)

```

```

; /*****
; /*
; /*      NAME:      NOP.CSM
; /*      VERSION:   1.0
; /*      DATE:      2 December 1983
; /*      MODULE NUMBER: 22
; /*      FUNCTION:   Provides delay required during loading of Direct
; /*                  Memory Access (DMA) chip's internal registers.
; /*      INPUTS:     NONE.
; /*      OUTPUTS:    NONE.
; /*      GLOBAL VARIABLES USED:  NONE.
; /*      GLOBAL VARIABLES CHANGED:  NONE.
; /*      FILES READ:  NONE.
; /*      FILES WRITTEN:  NONE.
; /*      MODULES CALLED:  NONE.
; /*      CALLING MODULES:  dma().
; /*
; /*      AUTHOR:     CAPTAIN WILLIAM H. LIEBER
; /*
; /*      HISTORY:    AFIT THESIS GE/EE/84D-71: Development of a Dedicated
; /*                  Speech Work Station.  Thesis Advisor: Major Larry Kizer.
; /*
; /*
; /*****/

```

INCLUDE bds.lib

FUNCTION nop

```

NOP          ;  ALLOWS SMALL DELAYS TO BE INSERTED INTO
NOP          ;  THE "C" PROGRAM.  REQUIRED IN DMA().
NOP
RET

```

ENDFUNCTION

```

; /*****
; /*
; /*      NAME:      WAIT.CSM
; /*      VERSION:   1.0
; /*      DATE:      2 December 1983
; /*      MODULE NUMBER: 23
; /*      FUNCTION:  Prevents CPU from executing SPEECH program statements*/
; /*                  until all data samples have been taken by analog() or played */
; /*                  out by digital(). Done by placing CPU in a tight loop and not */
; /*                  exiting the loop until an interrupt has occurred. The System */
; /*                  Timing Controller (STC) chip activates the interrupt signal */
; /*                  after all samples have been used.
; /*      INPUTS:    Interrupt line driven active by STC chip action.
; /*      OUTPUTS:   NONE.
; /*      GLOBAL VARIABLES USED:  NONE.
; /*      GLOBAL VARIABLES CHANGED:  NONE.
; /*      FILES READ:  NONE.
; /*      FILES WRITTEN:  NONE.
; /*      MODULES CALLED:  NONE.
; /*      CALLING MODULES:  analog(), digital().
; /*
; /*      AUTHOR:    CAPTAIN WILLIAM H. LIEBER
; /*
; /*      HISTORY:   AFIT THESIS GE/EE/84D-71: Development of a Dedicated */
; /*                  Speech Work Station. Thesis Advisor: Major Larry Kizer.
; /*
; /*
; /*      /*****

```

```
#INCLUDE bds.lib
```

```
FUNCTION wait
```

```

; /*****
; /*      CREATE STORAGE BYTE & SAVE REGISTERS
; /*
; /*      /*****
CHECK: DS      1      ; Byte to be checked during loop operation.
      PUSH     PSW    ; Save affected registers.
      PUSH     D      ;
      PUSH     H      ;

; /*****
; /*      LOAD INTERRUPT ROUTINE
; /*
; /*      /*****
MVI     L,38H      ; MVI data to Memory = 00110110B.
MVI     H,00H      ; Place "MVI data to M" command in memory
MVI     E,36H      ; location 38H.
MOV      M,E      ;

INR      L      ; Place data "01H" in memory location 39H.
MVI     E,01H      ;
MOV      M,E      ;

```

```

INR      L      ; RETURN = 11001001B.
MVI      E,0C9H ; Place "RET" command in memory location 3AH.
MOV      M,E    ;

;*****/
;*          WAIT UNTIL ALL SAMPLES HAVE BEEN TAKEN          */
;*****/
LXI      H,CHECK ; INITIALIZE: CHECK BYTE = 0
MVI      M,00H   ;          "A" REGISTER = 1
MVI      A,01H   ;
LOOP:    EI      ; Enable Interrupt.
        CMP      M      ; LOOP until CHECK BYTE = 1
        JNZ      LOOP   ;

;*****/
;*          DONE          */
;*****/
DI      ; Disable Interrupt.
POP      H      ; RESTORE affected registers.
POP      D      ;
POP      PSW     ;
RET      ; RETURN to calling program.

ENDFUNCTION

```

NOT TO  
SCALE

IC 13  
SN74LS32N

IC 6  
SN74LS125

IC 11  
SN74LS244

IC 12  
SN74LS244

IC 15  
SN74LS00

IC 2  
HM 6116P

IC 3  
HM 6116P

IC 7  
SN74LS74

IC 4  
SN74LS688

IC 5  
SN74LS125

IC 1  
SN74LS244

IC 10  
SN74LS688

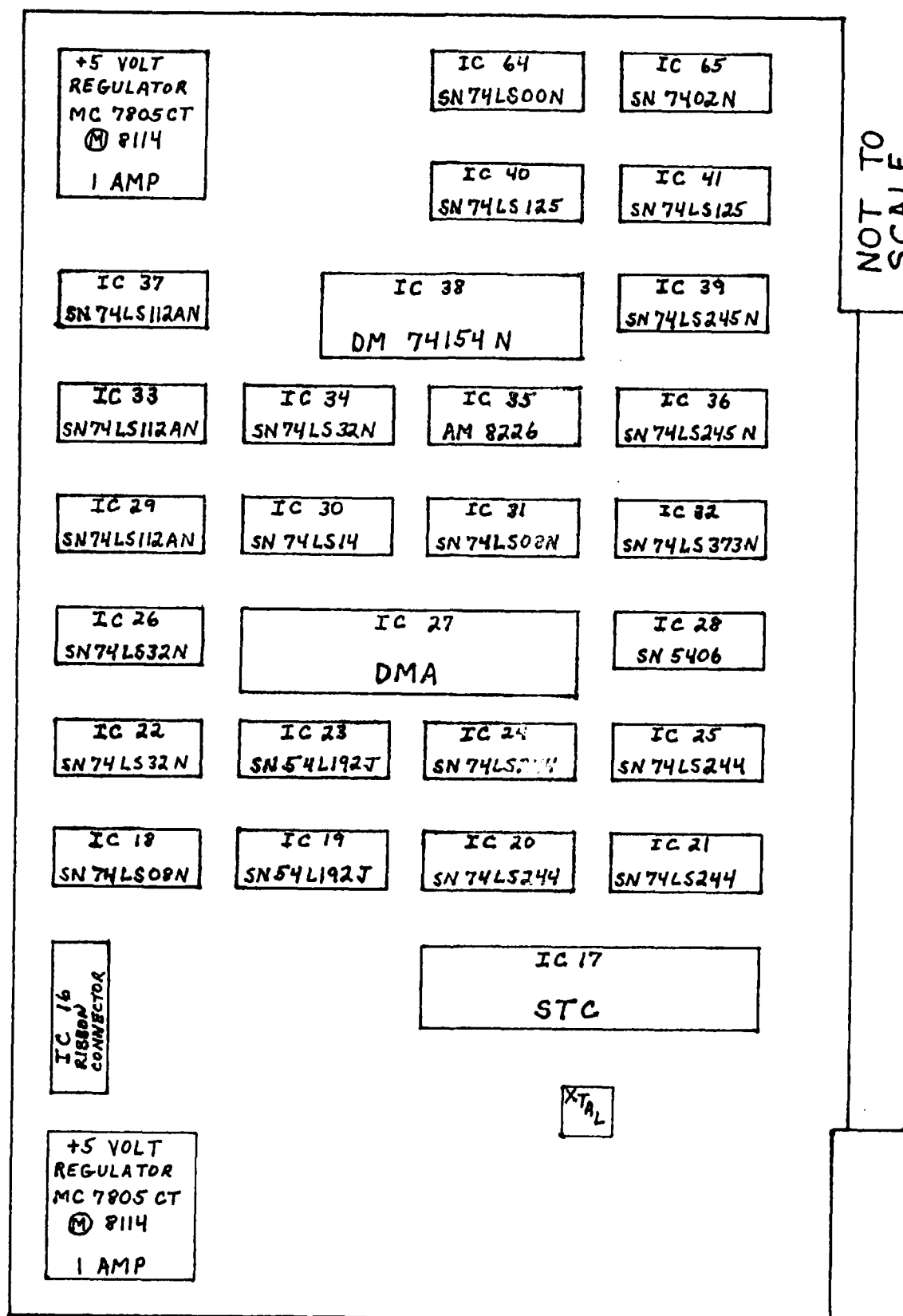
IC 9  
SN74LS688

IC 8  
SN74LS244

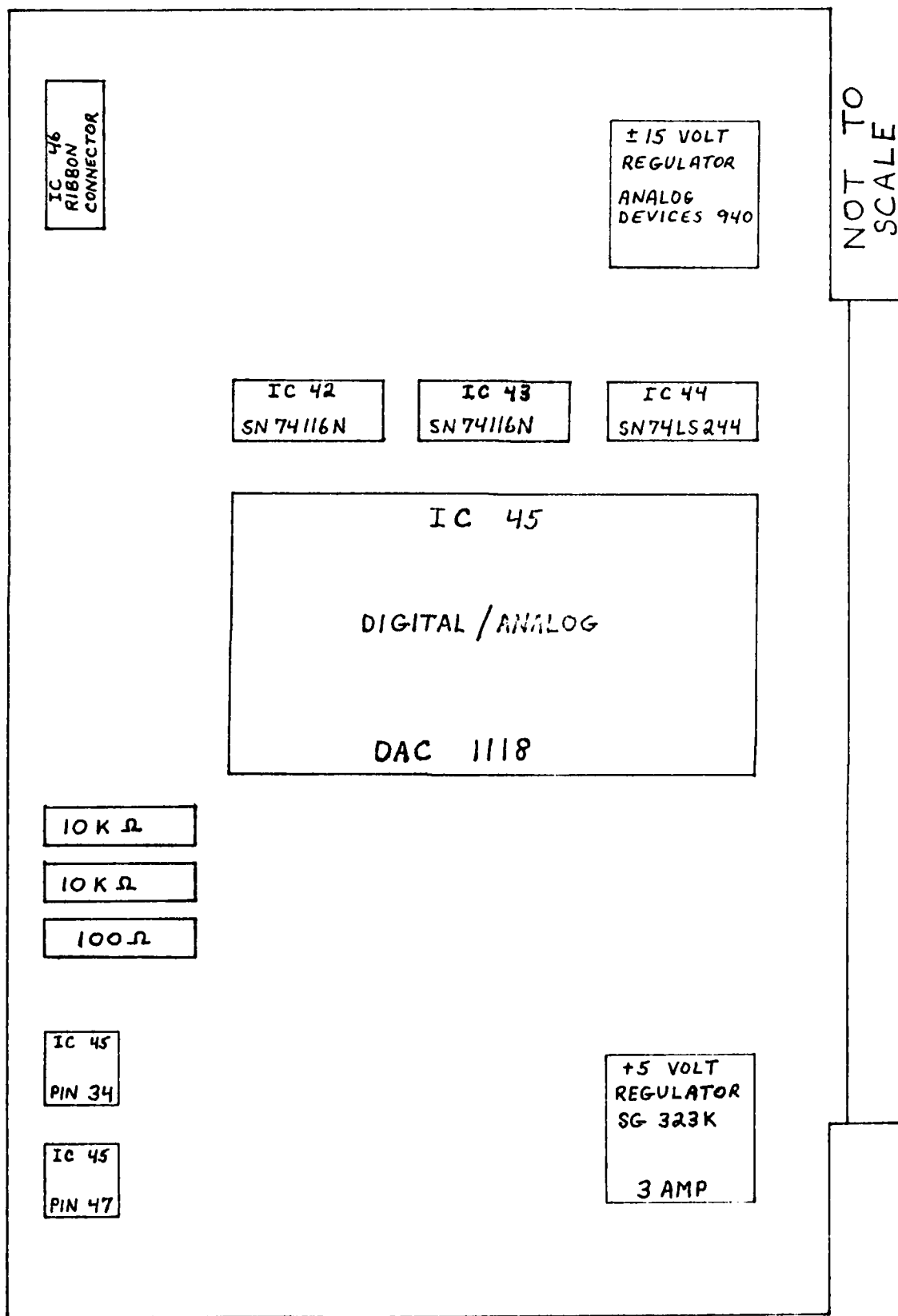
IC 14  
SN74LS14

+5 VOLT  
REGULATOR  
MC 7805 CT  
Ⓜ 8114  
1 AMP

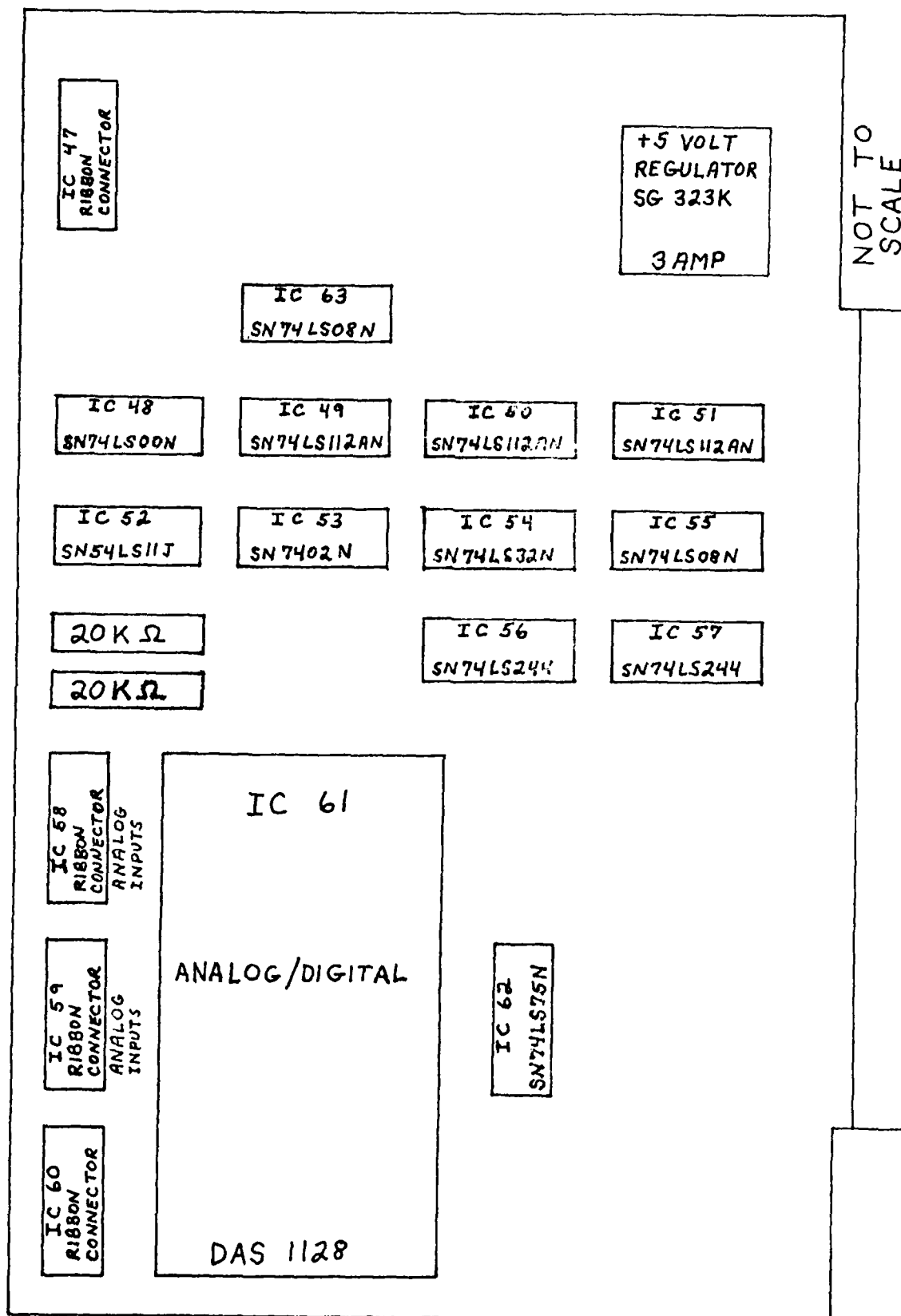
BOARD 1 - BOOT UP CONTROL



BOARD 2 - DIRECT MEMORY ACCESS



BOARD 3 - DIGITAL TO ANALOG CONVERSION

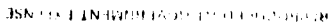


BOARD 4 - ANALOG TO DIGITAL CONVERSION

# LIST OF INTEGRATED CIRCUITS USED

IC #1	SN74LS244	Octal Bus Driver
IC #2	HM6116P	Memory Chip
IC #3	HM6116P	Memory Chip
IC #4	SN74LS688	8-Bit Compare
IC #5	SN74LS125	Quad Bus Driver
IC #6	SN74LS125	Quad Bus Driver
IC #7	SN74LS74	D Flip Flop
IC #8	SN74LS244	Octal Bus Driver
IC #9	SN74LS688	8-Bit Compare
IC #10	SN74LS688	8-Bit Compare
IC #11	SN74LS244	Octal Bus Driver
IC #12	SN74LS244	Octal Bus Driver
IC #13	SN74LS32N	2-Input OR
IC #14	SN74LS14	Hex Inverter
IC #15	SN74LS00	2-Input NAND
IC #16	Ribbon Interconnect	To Board Number 3
IC #17	AM9513	System Timing Chip
IC #18	SN74LS08N	2-Input AND
IC #19	SN54L192J	4-Bit Counter
IC #20	SN74LS244	Octal Bus Driver
IC #21	SN74LS244	Octal Bus Driver
IC #22	SN74LS32N	2-Input OR
IC #23	SN54L192J	4-Bit Counter
IC #24	SN74LS244	Octal Bus Driver
IC #25	SN74LS244	Octal Bus Driver
IC #26	SN74LS32N	2-Input OR
IC #27	AM9517	DIRECT MEMORY ACCESS
IC #28	SN5406	Hex Inverter (O.C.)
IC #29	SN74LS112AN	JK Flip Flop
IC #30	SN74LS14	Hex Inverter
IC #31	SN74LS08N	2-Input AND
IC #32	SN74LS373N	8-Bit Latch
IC #33	SN74LS112AN	JK Flip Flop
IC #34	SN74LS32N	2-Input OR
IC #35	AM8226	Bus Inverting
IC #36	SN74LS245N	Bus Bi-Directional
IC #37	SN74LS112AN	JK Flip Flop
IC #38	DM74154N	4 to 16 Decoder
IC #39	SN74LS245N	Bus Bi-Directional
IC #40	SN74LS125	Quad Bus Driver

IC #41	SN74LS125	Quad Bus Driver
IC #42	SN74116N	8-Bit Latch
IC #43	SN74116N	8-Bit Latch
IC #44	SN74LS244	Octal Bus Driver
IC #45	DAC 1118	DIGITAL to ANALOG
IC #46	Ribbon Interconnect	To Board Number 4
IC #47	Ribbon Interconnect	To Board Number 3
IC #48	SN74LS00N	2-Input NAND
IC #49	SN74LS112AN	JK Flip Flop
IC #50	SN74LS112AN	JK Flip Flop
IC #51	SN74LS112AN	JK Flip Flop
IC #52	SN74LS11J	3-Input AND
IC #53	SN7402N	2-Input NOR
IC #54	SN74LS32N	2-Input OR
IC #55	SN74LS08N	2-Input AND
IC #56	SN74LS244	Octal Bus Driver
IC #57	SN74LS244	Octal Bus Driver
IC #58	Ribbon Interconnect	To Analog Inputs
IC #59	Ribbon Interconnect	To Analog Inputs
IC #60	Ribbon Interconnect	To Board Number 2
IC #61	DAS 1128	ANALOG to DIGITAL
IC #62	SN74LS75N	Latch
IC #63	SN74LS08N	2-Input AND
IC #64	SN74LS00N	2-Input NAND
IC #65	SN7402N	2-Input NOR



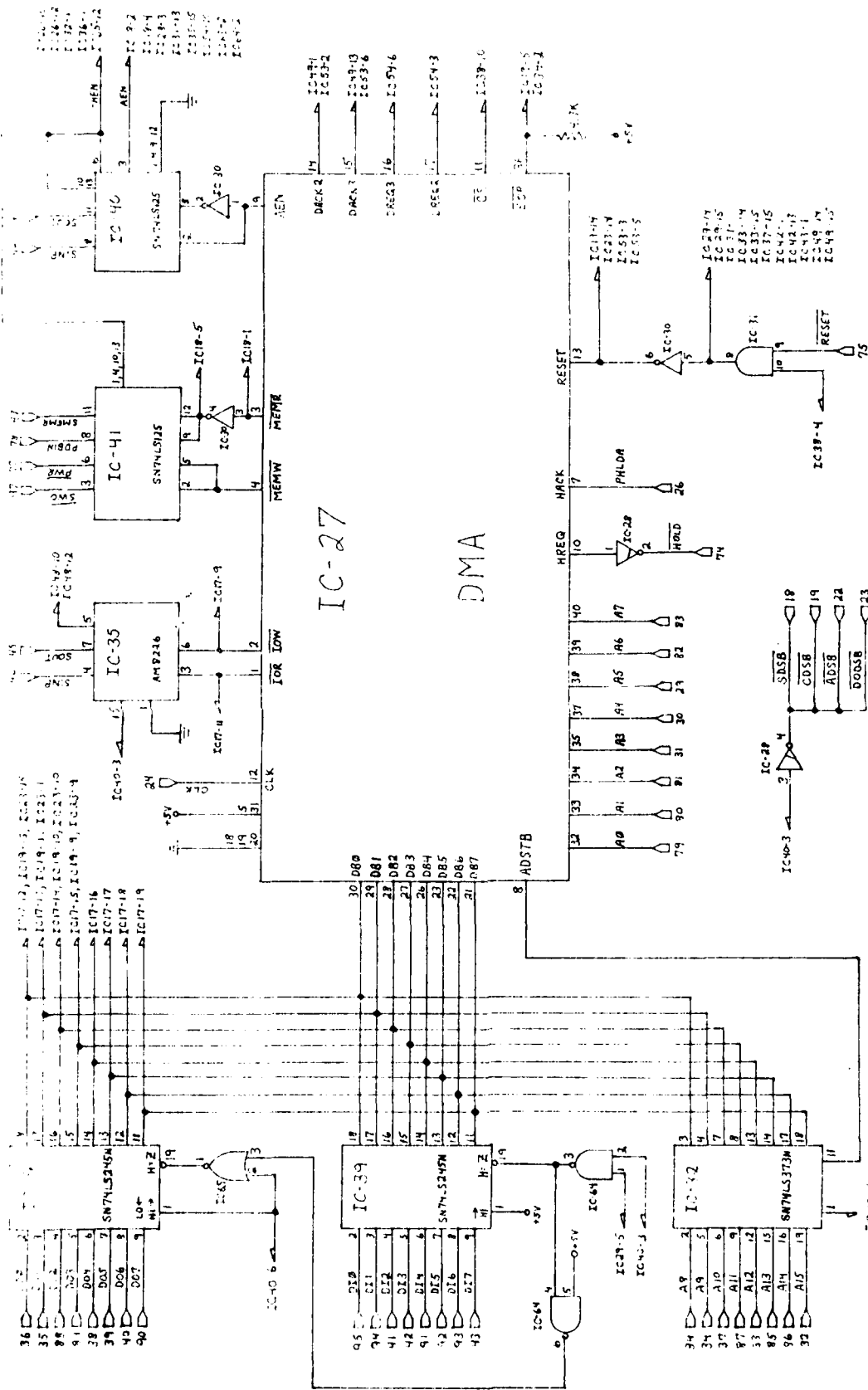
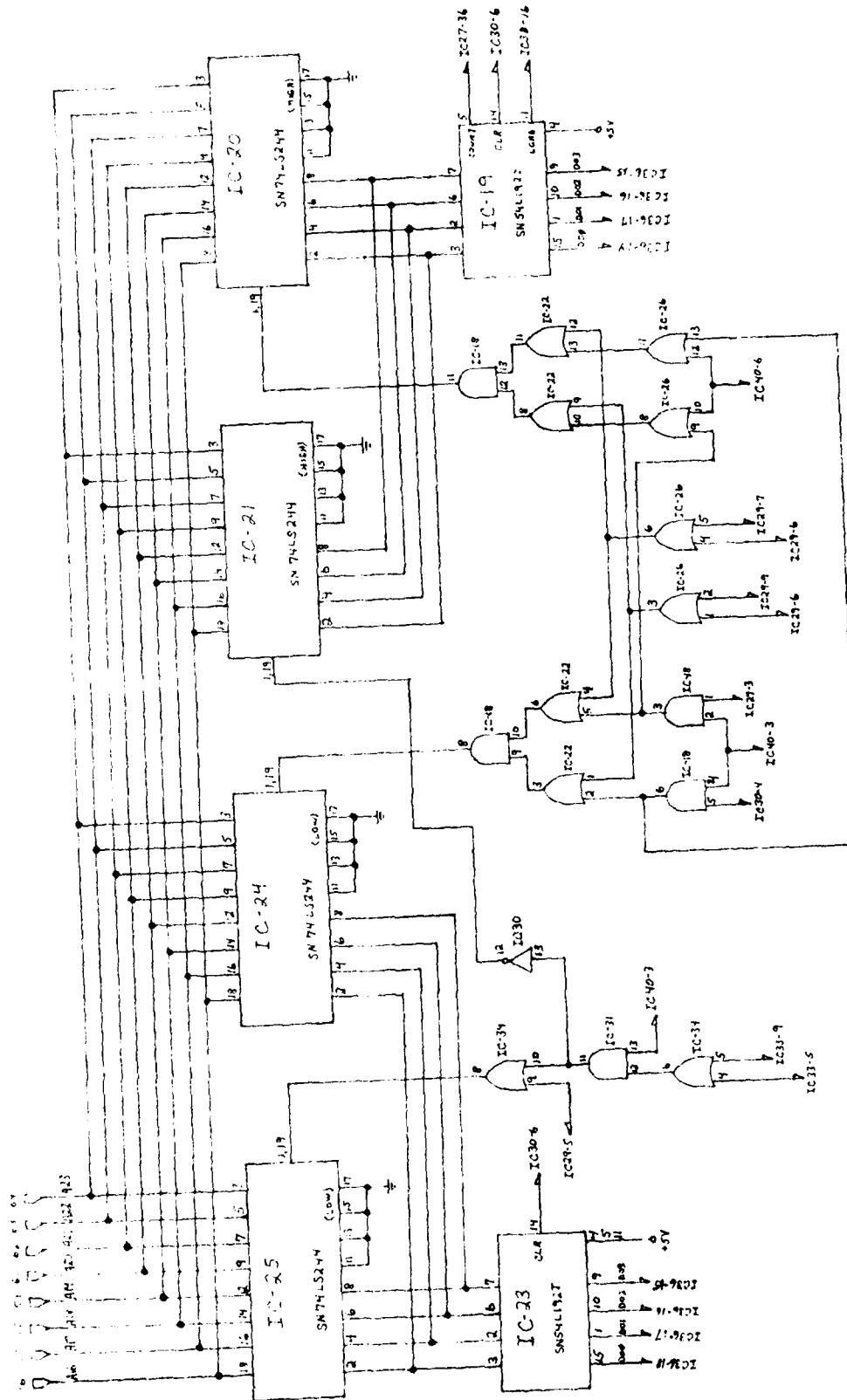
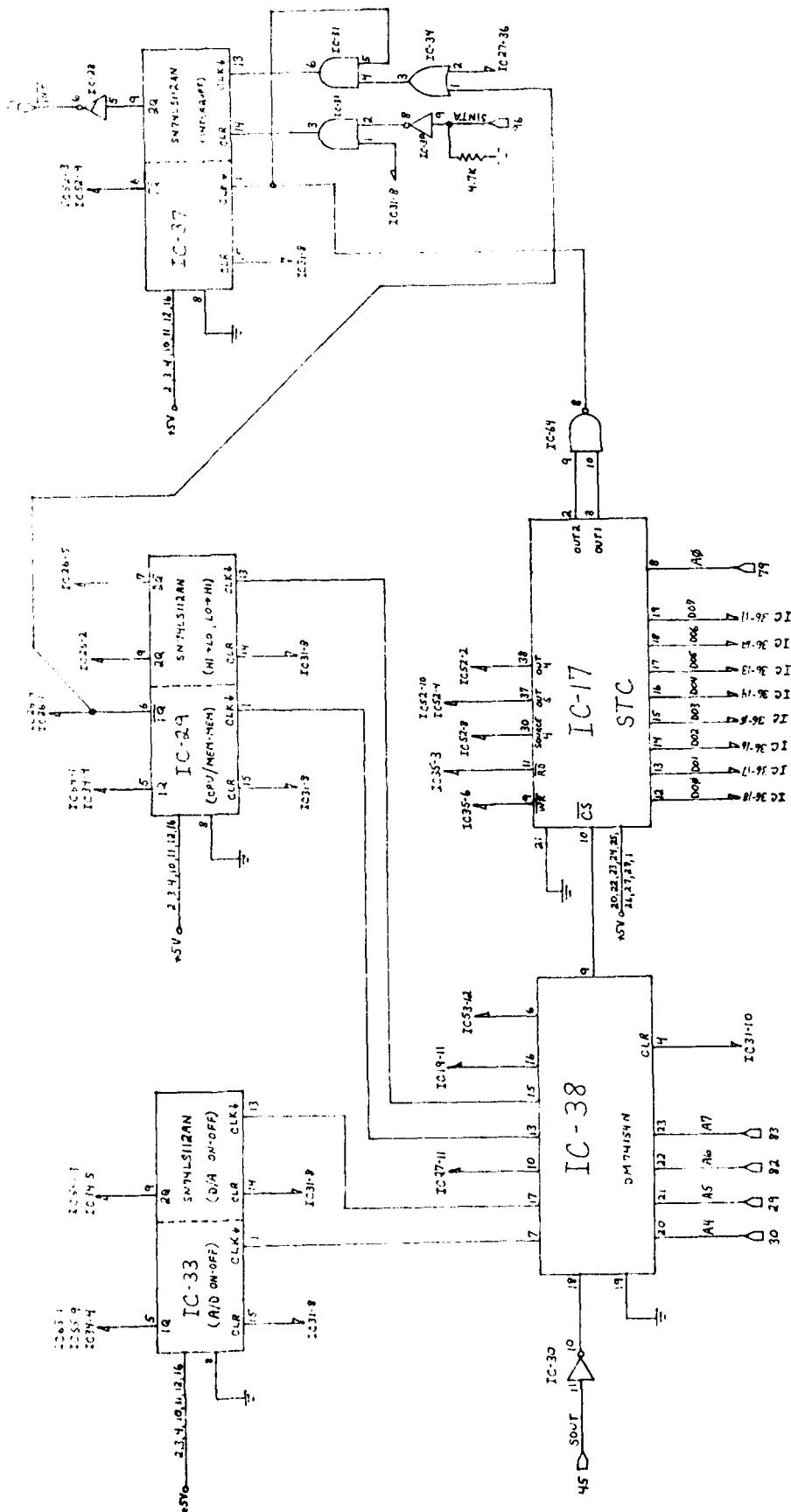


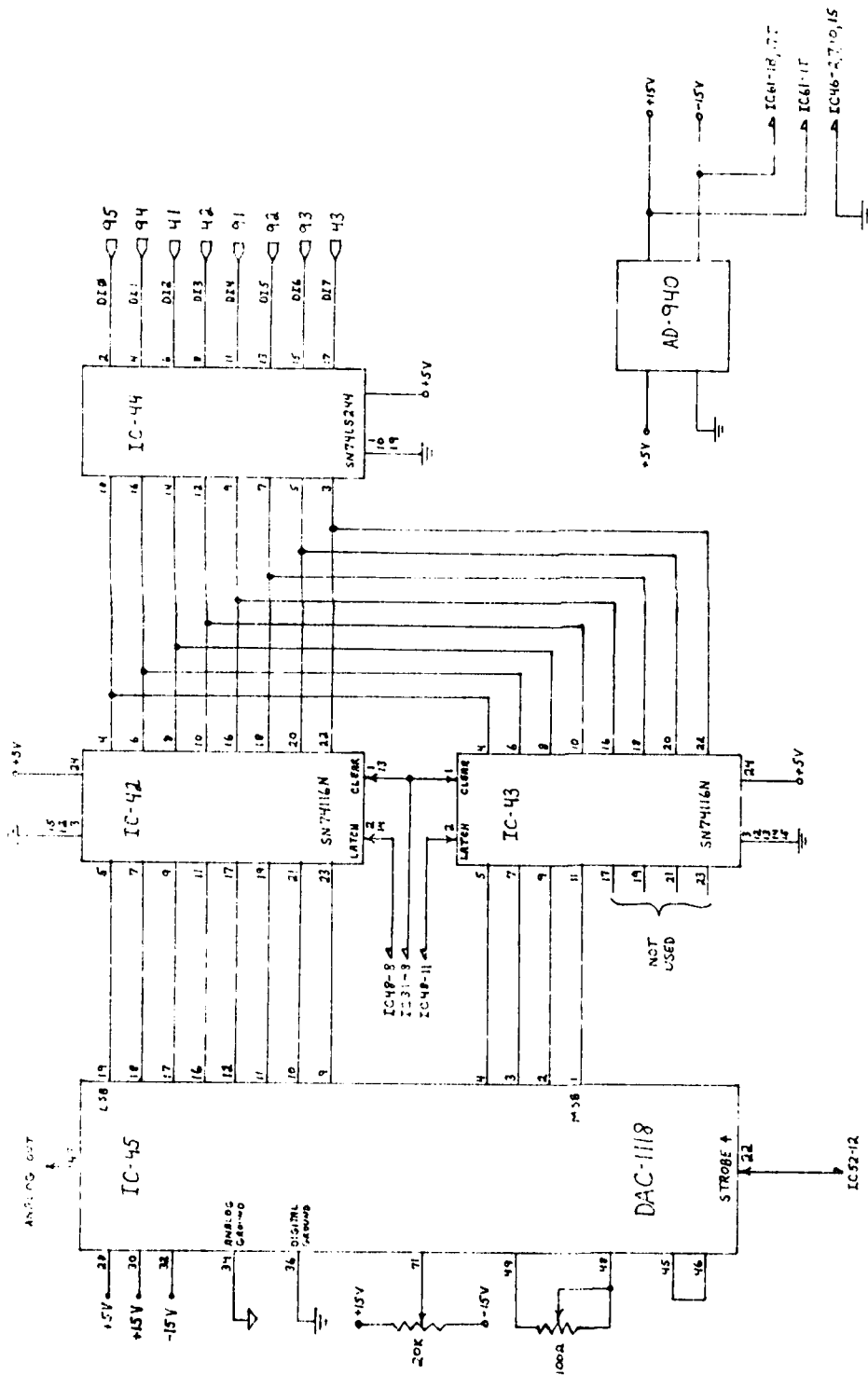
FIGURE 2 - DIRECT MEMORY ACCESS



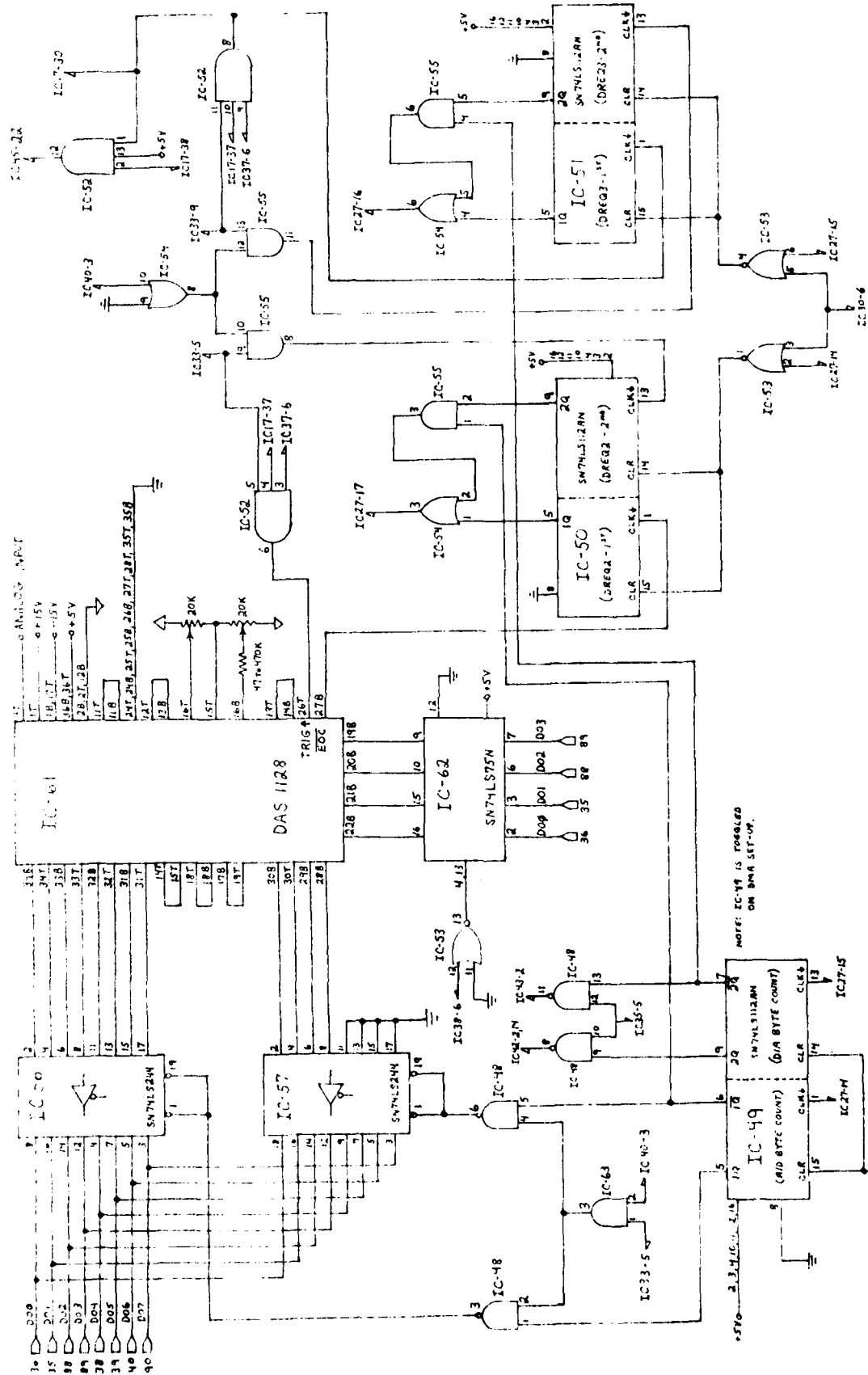
BOARD 2 - DIRECT MEMORY ACCESS



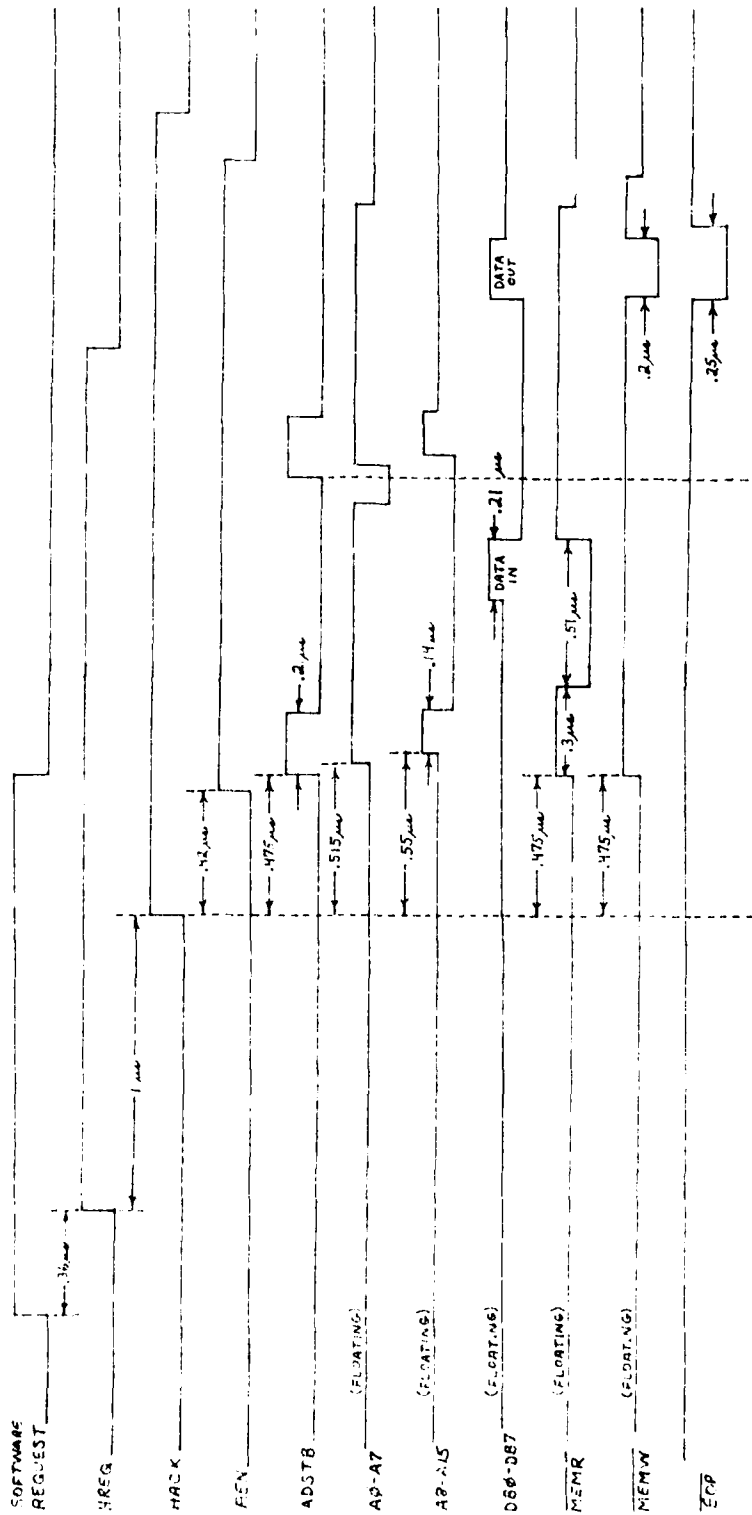
BOARD 2 - DIRECT MEMORY ACCESS



BOARD 1 - DIGITAL TO ANALOG CONVERSION

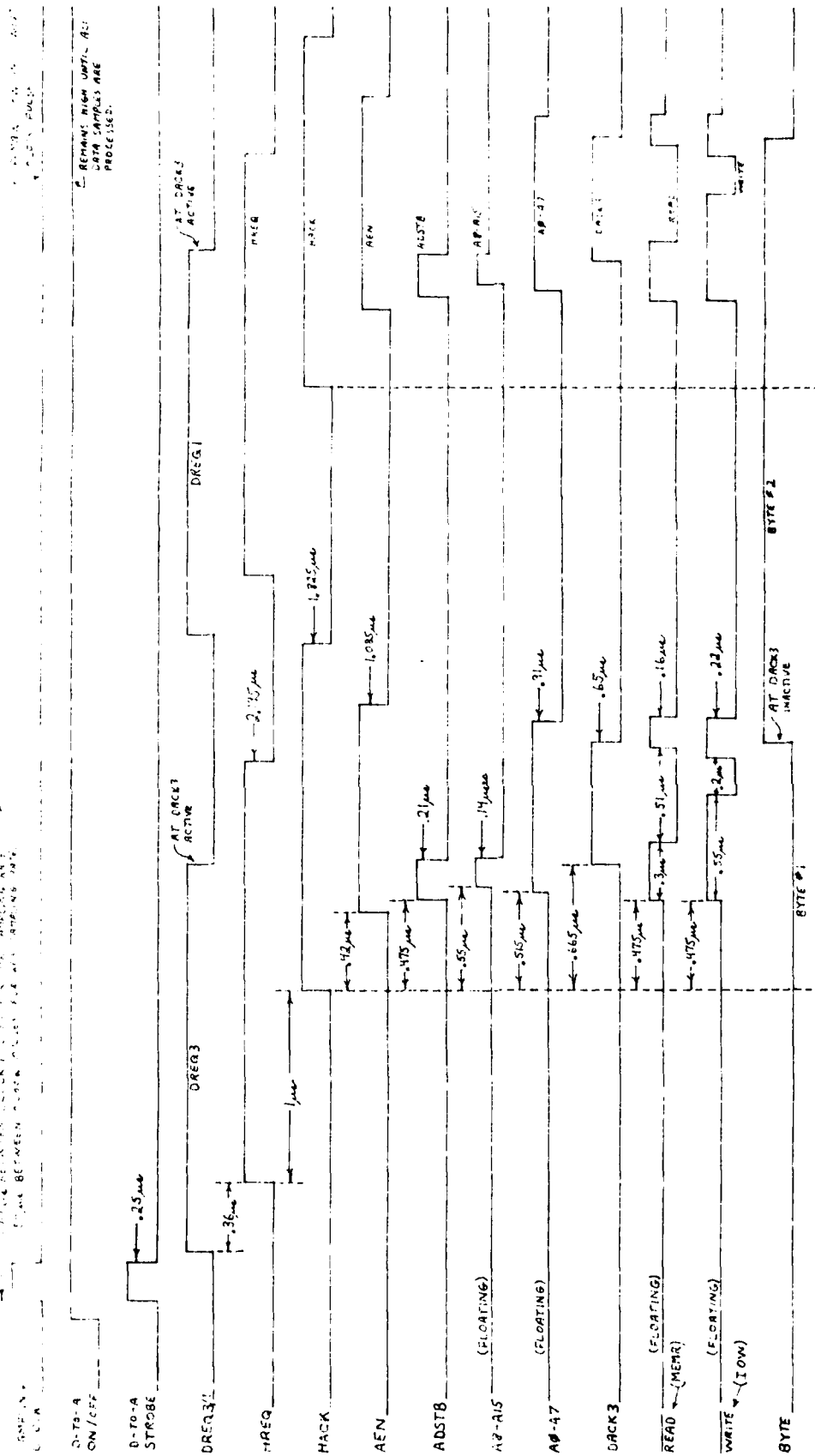


BOARD 1 - ANALOG TO DIGITAL CONVERSION

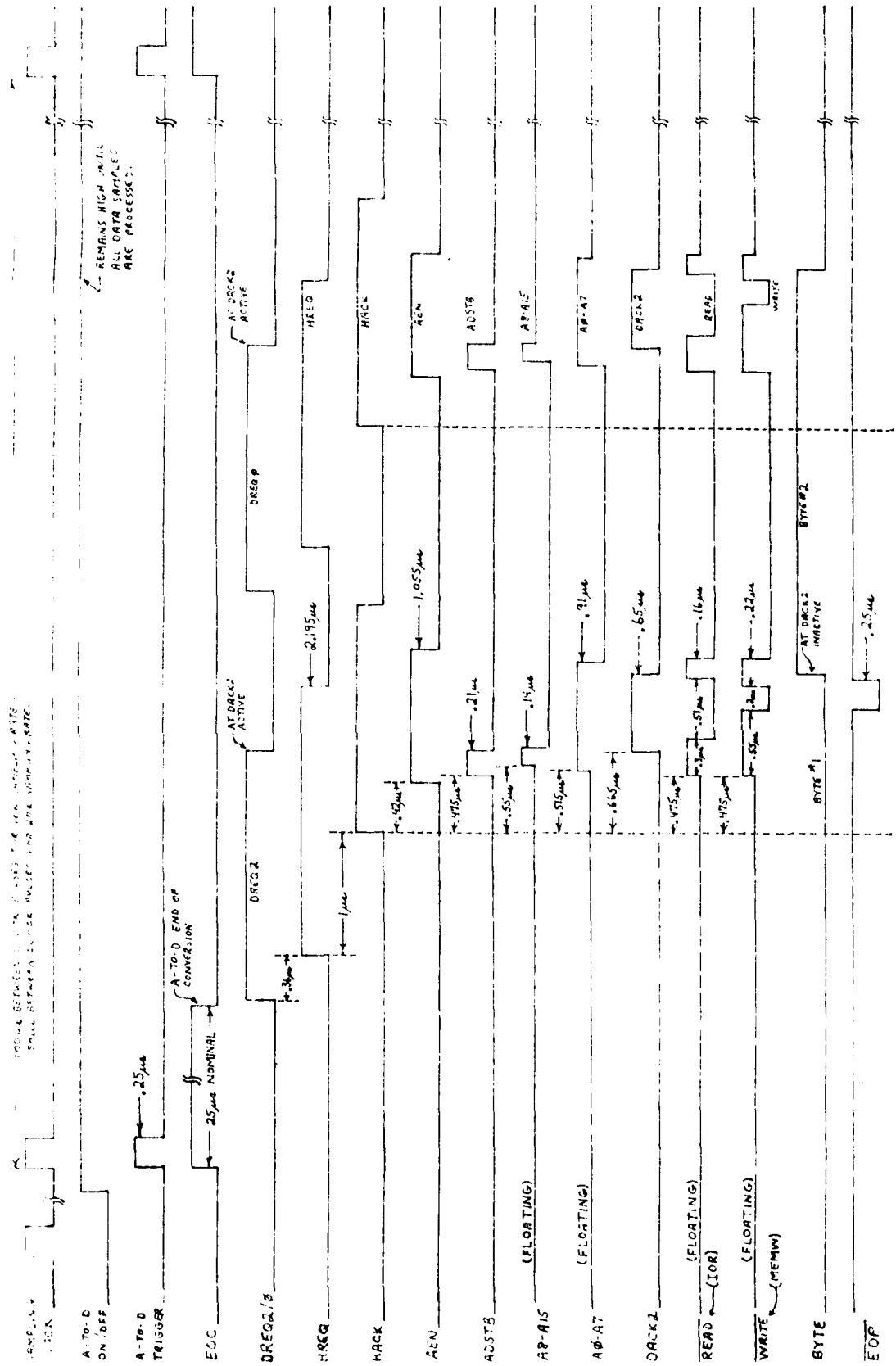


MEMORY TO MEMORY TRANSFER

E-1



DIGITAL TO ANALOG CONVERSION



NOISSEMO, TALLER, C. D. 2000

# Am9517A

## Multimode DMA Controller

### DISTINCTIVE CHARACTERISTICS

- Four independent DMA channels, each with separate registers for Mode Control, Current Address, Base Address, Current Word Count and Base Word Count.
- Transfer modes: Block, Demand, Single Word, Cascade
- Independent autoinitialization of all channels
- Memory-to-memory transfers
- Memory block initialization
- Address increment or decrement
- Master system disable
- Enable/disable control of individual DMA requests
- Directly expandable to any number of channels
- End of Process input for terminating transfers
- Software DMA requests
- Independent polarity control for DREQ and DACK signals
- Compressed timing option speeds transfers — up to 2M words/second
- +5 volt power supply
- Advanced N-channel silicon gate MOS technology
- 40 pin Hermetic DIP package
- 100 % MIL-STD-883 reliability assurance testing

### GENERAL DESCRIPTION

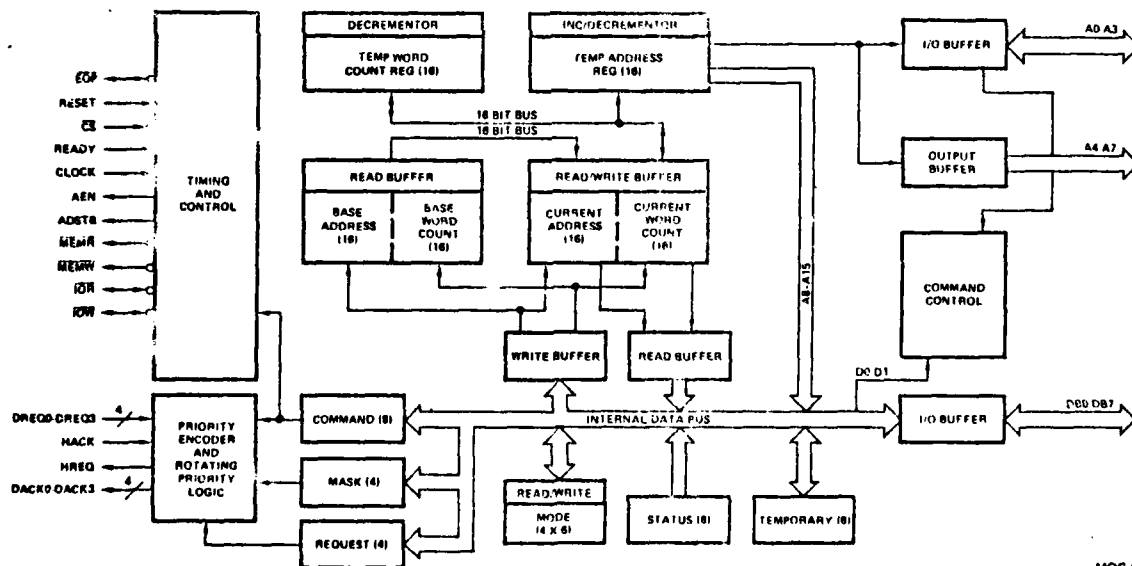
The Am9517A Multimode Direct Memory Access (DMA) Controller is a peripheral interface circuit for microprocessor systems. It is designed to improve system performance by allowing external devices to directly transfer information to or from the system memory. Memory-to-memory transfer capability is also provided. The Am9517A offers a wide variety of programmable control features to enhance data throughput and system optimization and to allow dynamic reconfiguration under program control.

The Am9517A is designed to be used in conjunction with an external 8-bit address register such as the Am74LS373. It contains four independent channels and may be expanded to any number of channels by cascading additional controller chips.

The three basic transfer modes allow programmability of the types of DMA service by the user. Each channel can be individually programmed to Autoinitialize to its original condition following an End of Process (EOP).

Each channel has a full 64K address and word count capability. An external EOP signal can terminate a DMA or memory-to-memory transfer. This is useful for block search or compare operations using external comparators or for intelligent peripherals to abort erroneous services.

### BLOCK DIAGRAM



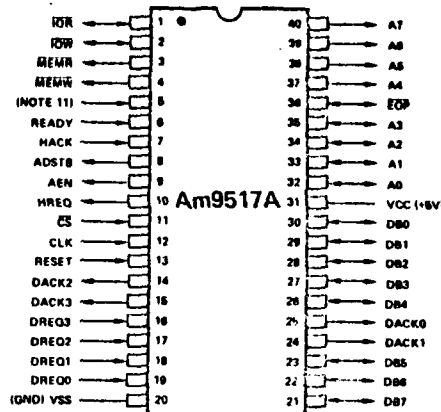
MOS-033

### ORDERING INFORMATION

Package Type	Ambient Temperature	Maximum Clock Frequency	
		3MHz	4MHz
Hermetic DIP/ Molded DIP	0°C ≤ T <sub>A</sub> ≤ +70°C	AM9517ADC/PC AM9517A-1DC/PC	AM9517A-4DC/PC
Hermetic DIP	-55°C ≤ T <sub>A</sub> ≤ +125°C	AM9517ADM	

## Am9517A

### CONNECTION DIAGRAM



Top View  
Pin 1 is marked for orientation.

Figure 1.

MOS 034

### INTERFACE SIGNAL DESCRIPTION

VCC: +5 Volt Supply  
VSS: Ground

#### CLK (Clock, Input)

This input controls the internal operations of the Am9517A and its rate of data transfers. The input may be driven at up to 3MHz for the standard Am9517A and up to 4MHz for the Am9517A-4.

#### CS (Chip Select, Input)

Chip Select is an active low input used to select the Am9517A as an I/O device during an I/O Read or I/O Write by the host CPU. This allows CPU communication on the data bus. During multiple transfers to or from the Am9517A by the host CPU, CS may be held low providing IOR or IOW is toggled following each transfer.

#### RESET (Reset, Input)

Reset is an asynchronous active high input which clears the Command, Status, Request and Temporary registers. It also clears the first/last flip/flop and sets the Mask register. Following a Reset the device is in the Idle cycle.

#### READY (Ready, Input)

Ready is an input used to extend the memory read and write pulses from the Am9517A to accommodate slow memories or I/O peripheral devices.

#### HACK (Hold Acknowledge, Input)

The active high Hold Acknowledge from the CPU indicates that control of the system buses has been relinquished.

#### DREQ0-DREQ3 (DMA Request, Input)

The DMA Request lines are individual asynchronous channel request inputs used by peripheral circuits to obtain DMA service. In Fixed Priority, DREQ0 has the highest priority and DREQ3 has the lowest priority. Polarity of DREQ is programmable. Reset initializes these lines to active high.

#### DB0-DB7 (Data Bus, Input/Output)

The Data Bus lines are bidirectional three-state signals connected to the system data bus. The outputs are enabled during the I/O Read by the host CPU, permitting the CPU to examine

the contents of an Address register, the Status register, the Temporary register or a Word Count register. The Data Bus is enabled to input data during a host CPU I/O write, allowing the CPU to program the Am9517A control registers. During DMA cycles the most significant eight bits of the address are output onto the data bus to be strobed into an external latch by ADSTB. In memory-to-memory operations data from the source memory location comes into the Am9517A's Temporary register on the read-from-memory half of the operation. On the write-to-memory half of the operation, the data bus outputs the Temporary register data into the destination memory location.

#### IOR (I/O Read, Input/Output)

I/O Read is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to read the control registers. In the Active cycle, it is an output control signal used by the Am9517A to access data from a peripheral during a DMA Write transfer.

#### IOW (I/O Write, Input/Output)

I/O Write is a bidirectional active low three-state line. In the Idle cycle it is an input control signal used by the CPU to load information into the Am9517A. In the Active cycle it is an output control signal used by the Am9517A to load data to the peripheral during a DMA Read transfer.

Write operations by the CPU to the Am9517A require a rising WR edge following each data byte transfer. It is not sufficient to hold the IOW pin low and toggle CS.

#### EOP (End of Process, Input/Output)

EOP is an active low bidirectional open-drain signal providing information concerning the completion of DMA service. When a channel's Word Count goes to zero, the Am9517A pulses EOP low to provide the peripheral with a completion signal. EOP may also be pulled low by the peripheral to cause premature completion. The reception of EOP, either internal or external, causes the currently active channel to terminate the service, to set its TC bit in the Status register and to reset its request bit. If Autoinitialization is selected for the channel, the current registers will be updated from the base registers. Otherwise the channel's mask bit will be set and the register contents will remain unaltered.

During memory-to-memory transfers, the TC for channels with an active I/O pin is required. V<sub>CC</sub> is required. EOP pin can not be used as an output address.

#### A0-A3 (Address, Input)

The four least significant address signals. During DMA transfer, host CPU to load active, they are output address.

#### A4-A7 (Address, Input)

The four most significant address signals and provide four address during DMA service.

#### HREQ (Hold Request, Input)

The Hold Request signal is used for control of the system buses. DREQs cause the

#### DACK0-DACK3 (DMA Acknowledge, Input)

The DMA Acknowledge signals. Many systems the DACK will be active low. DMA is in control, programmable. Reset

#### AEN (Address Enable, Input)

Address Enable is an active low signal used to enable the system bus during DMA transfer. It is used as program input during DMA transfer.

#### ADSTB (Address Strobe, Input)

The active high Address Strobe signal is used to address byte from the data bus.

#### MEMR (Memory Read, Input)

The Memory Read signal is used to access data from memory-to-peripheral.

N	Base Address Register
0	Base Word Count
1	Current Address
2	Current Word Count
3	Temporary Address
4	Temporary Word Count
5	Status Register
6	Command Register
7	Temporary Register
8	Mode Register
9	Mask Register
10	Request Register

Figure 2.

During memory-to-memory transfers,  $\overline{EOP}$  will be output when the TC for channel 1 occurs.  $\overline{EOP}$  always applies to the channel with an active DACK; external  $\overline{EOPS}$  are disregarded in DACK0-DACK3 are all inactive.

Because  $\overline{EOP}$  is an open-drain signal, an external pullup resistor is required. Values of 3.3K or 4.7K are recommended; the  $\overline{EOP}$  pin can not sink the current passed by a 1K pullup.

#### A0-A3 (Address, Input/Output)

The four least significant address lines are bidirectional 3-state signals. During DMA Idle cycles they are inputs and allow the host CPU to load or read control registers. When the DMA is active, they are outputs and provide the lower 4-bits of the output address.

#### A4-A7 (Address, Output)

The four most significant address lines are three-state outputs and provide four bits of address. These lines are enabled only during DMA service.

#### HREQ (Hold Request, Output)

The Hold Request to the CPU is used by the DMA to request control of the system bus. Software requests or unmasked DREQs cause the Am9517A to issue HREQ.

#### DACK0-DACK3 (DMA Acknowledge, Output)

The DMA Acknowledge lines indicate that a channel is active. In many systems they will be used to select a peripheral. Only one DACK will be active at a time and none will be active unless the DMA is in control of the bus. The polarity of these lines is programmable. Reset initializes them to active-low.

#### AEN (Address Enable, Output)

Address Enable is an active high signal used to disable the system bus during DMA cycles to enable the output of the external latch which holds the upper byte of the address. Note that during DMA transfers HACK and AEN should be used to de-select all other I/O peripherals which may erroneously be accessed as programmed I/O during the DMA operation. The Am9517A automatically deselected itself by disabling the  $\overline{CS}$  input during DMA transfers.

#### ADSTB (Address Strobe, Output)

The active high Address Strobe is used to strobe the upper address byte from DB0-DB7 into an external latch.

#### MEMR (Memory Read, Output)

The Memory Read signal is an active low three-state output used to access data from the selected memory location during a memory-to-peripheral or a memory-to-memory transfer.

Name	Size	Number
Base Address Registers	16 bits	4
Base Word Count Registers	16 bits	4
Current Address Registers	16 bits	4
Current Word Count Registers	16 bits	4
Temporary Address Register	16 bits	1
Temporary Word Count Register	16 bits	1
Status Register	8 bits	1
Command Register	8 bits	1
Temporary Register	8 bits	1
Mode Registers	6 bits	4
Mask Register	4 bits	1
Request Register	4 bits	1

Figure 2. Am9517A Internal Registers.

#### MEMW (Memory Write, Output)

The Memory Write signal is an active low three-state output used to write data to the selected memory location during a peripheral-to-memory or a memory-to-memory transfer.

### FUNCTIONAL DESCRIPTION

The Am9517A block diagram includes the major logic blocks and all of the internal registers. The data interconnection paths are also shown. Not shown are the various control signals between the blocks. The Am9517A contains 344 bits of internal memory in the form of registers. Figure 2 lists these registers by name and shows the size of each. A detailed description of the registers and their functions can be found under Register Description.

The Am9517A contains three basic blocks of control logic. The Timing Control block generates internal timing and external control signals for the Am9517A. The Program Command Control block decodes the various commands given to the Am9517A by the microprocessor prior to servicing a DMA Request. It also decodes each channel's Mode Control word. The Priority Encoder block resolves priority contention among DMA channels requesting service simultaneously.

The Timing Control block derives internal timing from the clock input. In Am9080A systems this input will usually be the  $\phi 2$  TTL clock from an Am8224. However, any appropriate system clock will suffice.

#### DMA Operation

The Am9517A is designed to operate in two major cycles. These are called Idle and Active cycles. Each device cycle is made up of a number of states. The Am9517A can assume seven separate states, each composed of one full clock period. State 1 (S1) is the inactive state. It is entered when the Am9517A has no valid DMA requests pending. While in S1, the DMA controller is inactive but may be in the Program Condition, being programmed by the processor. State 0 (S0) is the first state of a DMA service. The Am9517A has requested a hold but the processor has not yet returned an acknowledge. An acknowledge from the CPU will signal that transfers may begin. S1, S2, S3 and S4 are the working states of the DMA service. If more time is needed to complete a transfer than is available with normal timing wait states (SW) can be inserted before S4 by the use of the Ready line on the Am9517A.

Memory-to-memory transfers require a read-from and a write-to-memory to complete each transfer. The states, which resemble the normal working states, use two digit numbers for identification. Eight states are required for each complete transfer. The first four states (S11, S12, S13, S14) are used for the read-from-memory half and the last four states (S21, S22, S23 and S24) for the write-to-memory half of the transfer. The Temporary Data register is used for intermediate storage of the memory byte.

#### IDLE Cycle

When no channel is requesting service, the Am9517A will enter the Idle cycle and perform "S1" states. In this cycle the Am9517A will sample the DREQ lines every clock cycle to determine if any channel is requesting a DMA service. The device will also sample  $\overline{CS}$ , looking for an attempt by the microprocessor to write or read the internal registers of the Am9517A. When  $\overline{CS}$  is low and HACK is low the Am9517A enters the Program Condition. The CPU can now establish, change or inspect the internal definition of the part by reading from or writing to the internal registers. Address lines A0-A3 are inputs to the device and select which registers will be read or written. The  $\overline{IOR}$  and  $\overline{IOW}$  lines are used to select and time reads or writes. Due to the

MOS-034

Status register, the Data Bus is O write, allowing the registers. During DMA address are output latch by ADSTB. the source memory rary register on the the write-to-memory the Temporary register.

state line. In the Idle the CPU to read the output control signal peripheral during a

state line. In the Idle the CPU to load information cycle it is an output load data to the

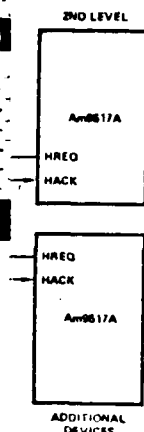
7A require a rising it is not sufficient to

in signal providing 7A service. When a 9517A pulses  $\overline{EOP}$  in signal,  $\overline{EOP}$  may premature completion, external, causes the ice, to set its TC bit st bit. If Autoinitial- nt registers will be he channel's mask remain unaltered.



the outputs of the  
517 will respond  
keep 2 will be

aded into an initial  
This forms a two  
e added at the sec-  
s of the first level,  
cascading into the  
g a third level.



As.

per free dif-  
Wn and Verify.  
to the memory by  
move data from  
A and IOW. Verify  
A operates as in  
es, responding to  
ontrol lines remain

es a block move  
ed from one mem-  
in the Command  
1 will operate as  
nel 0 forms the  
sination address.  
to-memory trans-  
fer for channel 0.  
emory-to-memory.  
source address, a  
ck of memory.

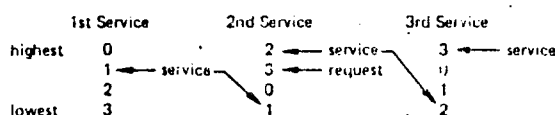
to-memory opera-  
1 be masked out.  
e initialized to the  
puts will be active

IP signals during  
rs in block search  
e service when a  
ory transfers may

**Autoinitialize:** By programming a bit in the Mode register a channel may be set up for an Autoinitialize operation. During Autoinitialization, the original values of the Current Address and Current Word Count registers are automatically restored from the Base Address and Base Word Count registers of that channel following EOP. The base registers are loaded simultaneously with the current registers by the microprocessor and remain unchanged throughout the DMA service. The mask bit is not set by EOP when the channel is in Autoinitialize. Following Autoinitialize the channel is ready to repeat its service without CPU intervention.

**Priority:** The Am9517A has two types of priority encoding available as software selectable options. The first is Fixed Priority which fixes the channels in priority order based upon the descending value of their number. The channel with the lowest priority is 3 followed by 2, 1 and the highest priority channel, 0.

The second scheme is Rotating Priority. The last channel to get service becomes the lowest priority channel with the others rotating accordingly. With Rotating Priority in a single chip DMA system, any device requesting service is guaranteed to be recognized after no more than three higher priority services have occurred. This prevents any one channel from monopolizing the system.



The priority encoder selects the highest priority channel requesting service on each active-going HACK edge. Once a channel is started, its operation will not be suspended if a request is received by a higher priority channel. The high priority channel will only gain control after the lower priority channel releases HREQ. When control is passed from one channel to another, the CPU will always gain bus control. This ensures generation of rising HACK edge to be used to initiate selection of the new highest-priority requesting channel.

**Compressed Timing:** In order to achieve even greater throughput where system characteristics permit, the Am9517A can compress the transfer time to two clock cycles. From Timing Diagram 3 it can be seen that state S3 is used to extend the access time of the read pulse. By removing state S3 the read pulse width is made equal to the write pulse width and a transfer consists only of state S2 to change the address and state S4 to perform the read/write. S1 states will still occur when A8-A15 need updating (see Address Generation). Timing for compressed transfers is found in Timing Diagram 6.

**Address Generation:** In order to reduce pin count, the Am9517A multiplexes the eight higher order address bits on the data lines. State S1 is used to output the higher order address

bits to an external latch from which they may be placed on the address bus. The falling edge of Address Strobe (ADS1B) is used to load these bits from the data lines to the latch. Address Enable (AEN) is used to enable the bits onto the address bus through a 3-state enable. The lower order address bits are output by the Am9517A directly. Lines A0-A7 should be connected to the address bus. Timing Diagram 3 shows the time relationships between CLK, AEN, ADS1B, DB0-DB7 and A0-A7.

During Block and Demand Transfer mode services which include multiple transfers, the addresses generated will be sequential. For many transfers the data held in the external address latch will remain the same. This data need only change when a carry or borrow from A7 to A8 takes place in the normal sequence of addresses. To save time and speed transfers, the Am9517A executes S1 states only when updating of A8-A15 in the latch is necessary. This means for long services, S\* states may occur only once every 256 transfers, a savings of 255 clock cycles for each 256 transfers.

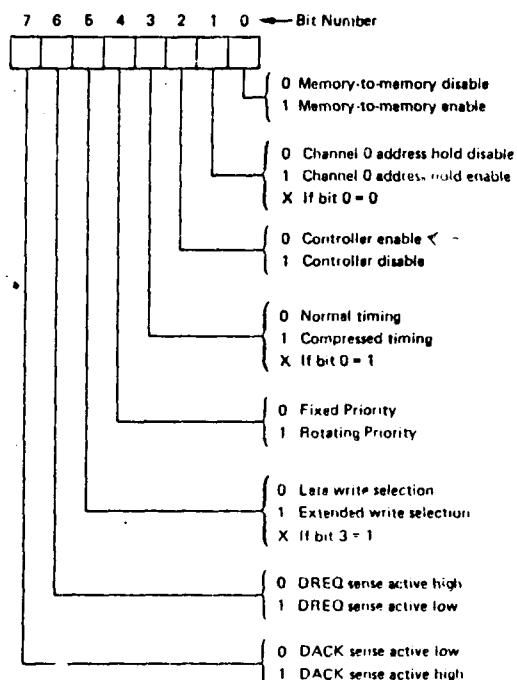
#### REGISTER DESCRIPTION

**Current Address Register:** Each channel has a 16-bit Current Address register. This register holds the value of the address used during DMA transfers. The address is automatically incremented or decremented after each transfer and the intermediate values of the address are stored in the Current Address register during the transfer. This register is written or read by the microprocessor in successive 8-bit bytes. It may also be reinitialized by an Autoinitialize back to its original value. Autoinitialization takes place only after an EOP.

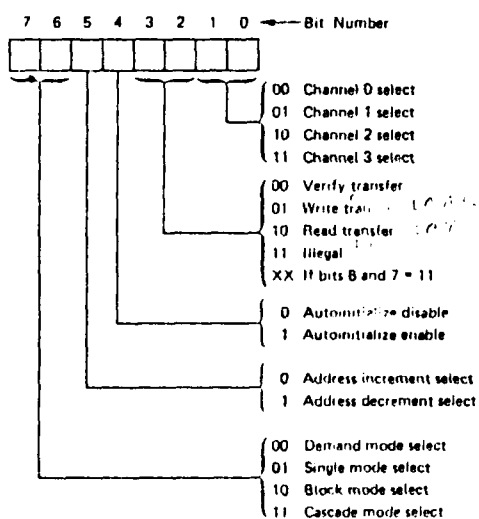
**Current Word Count Register:** Each channel has a 16-bit Current Word Count register. This register should be programmed with, and will return on a CPU read, a value one less than the number of words to be transferred. The word count is decremented after each transfer. The intermediate value of the word count is stored in the register during the transfer. When the value in the register goes to zero, a TC will be generated. This register is loaded or read in successive 8-bit bytes by the microprocessor in the Program Condition. Following the end of a DMA service it may also be reinitialized by an Autoinitialize back to its original value. Autoinitialize can occur only when an EOP occurs. Note that the contents of the Word Count register will be FFFF (hex) following an internally generated EOP.

**Base Address and Base Word Count Registers:** Each channel has a pair of Base Address and Base Word Count registers. These 16-bit registers store the original values of their associated current registers. During Autoinitialize these values are used to restore the current registers to their original values. The base registers are written simultaneously with their corresponding current register in 8-bit bytes during DMA programming by the microprocessor. Accordingly, writing to these registers when intermediate values are in the Current registers will overwrite the intermediate values. The Base registers cannot be read by the microprocessor.

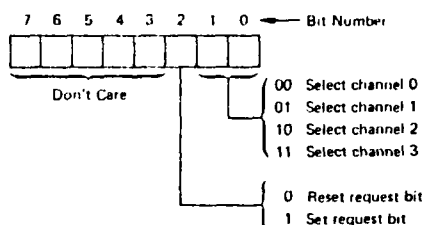
**Command Register:** This 8-bit register controls the operation of the Am9517A. It is programmed by the microprocessor in the Program Condition and is cleared by Reset. The following table lists the function of the command bits. See Figure 4 for address coding.



**Mode Register:** Each channel has a 6-bit Mode register associated with it. When the register is being written to by the microprocessor in the Program Condition, bits 0 and 1 determine which channel Mode register it is to be written.

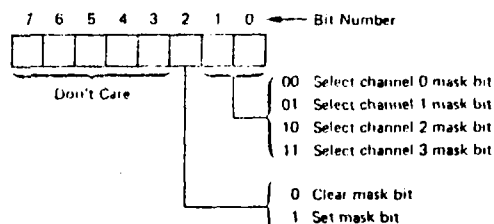


**Request Register:** The Am9517A can respond to requests for DMA service which are initiated by software as well as by a DREQ. Each channel has a request bit associated with it in the 4-bit Request register. These are nonmaskable and subject to prioritization by the Priority Encoder network. Each register bit is set or reset separately under software control or is cleared upon generation of a TC or external EOP. The entire register is cleared by a Reset. To set or reset a bit, the software loads the proper form of the data word. See Figure 4 for address coding.

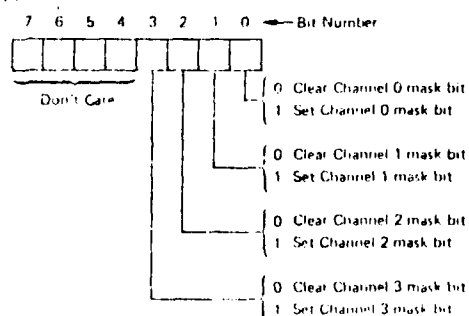


Software requests will be serviced only if the channel is in Block mode. When initiating a memory-to-memory transfer, the software request for channel 0 should be set.

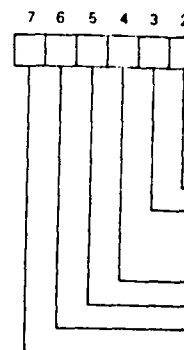
**Mask Register:** Each channel has associated with it a mask bit which can be set to disable the incoming DREQ. Each mask bit is set when its associated channel produces an EOP if the channel is not programmed for Autoinitialize. Each bit of the 4-bit Mask register may also be set or cleared separately under software control. The entire register is also set by a Reset. This disables all DMA requests until a clear Mask register instruction allows them to occur. The instruction to separately set or clear the mask bit is similar in form to that used with the Request register. See Figure 4 for instruction addressing.



All four bits of the Mask Register may also be written with a single command.



**Status Register:** The Am9517A by the micro have reached a terminal DMA requests. Bits 0 that channel, including are cleared by Reset or whenever their correspo



in re... requests for  
software... well as by a  
it associated with it in the  
maskable and subject to  
network. Each register bit is  
control or is cleared upon  
2. The entire register is  
bit, the software loads the  
are 4 for address coding.

Bit Number

- 00 Select channel 0
- 01 Select channel 1
- 10 Select channel 2
- 11 Select channel 3

- 0 Reset request bit
- 1 Set request bit

if the channel is in Block  
memory transfer, the  
be set.

ociated with it a mask bit  
g DREQ. Each mask bit  
roduces an EOP if the  
alize. Each bit of the 4-bit  
eared separately under  
also set by a Reset. This  
Mask register instruction  
se... set or clear  
use... the Request  
dressing.

Bit Number

- Select channel 0 mask bit
- Select channel 1 mask bit
- Select channel 2 mask bit
- Select channel 3 mask bit

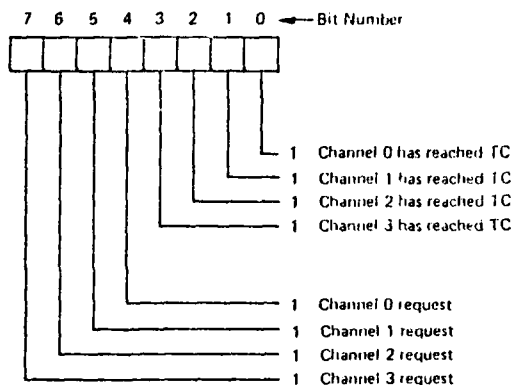
- Clear mask bit
- Set mask bit

also be written with a

Bit Number

- ear Channel 0 mask bit
- t Channel 0 mask bit
- ear Channel 1 mask bit
- t Channel 1 mask bit
- ear Channel 2 mask bit
- t Channel 2 mask bit
- ear Channel 3 mask bit
- t Chan... 3 mask bit

**Status Register:** The Status registers may be read out of the Am9517A by the microprocessor. It indicates which channels have reached a terminal count and which channels have pending DMA requests. Bits 0-3 are set each time a TC is reached by that channel, including after each Autoinitialization. These bits are cleared by Reset and each Status Read. Bits 4-7 are set whenever their corresponding channel is requesting service.



**Temporary Register:** The Temporary register is used to hold data during memory-to-memory transfers. Following the completion of the transfers, the last word moved can be read by the microprocessor in the Program Condition. The Temporary register always contains the last byte transferred in the previous memory-to-memory operation, unless cleared by a Reset.

**Software Commands:** There are two special software commands which can be executed in the Program Condition. They do not depend on any specific bit pattern on the data bus. The two software commands are:

**Clear First/Last Flip/Flop:** This command may be issued prior to writing or reading Am9517A address or word count information. This initializes the flip/flop to a known state so that subsequent accesses to register contents by the microprocessor will address lower and upper bytes in the correct sequence.

**Master Clear:** This software instruction has the same effect as the hardware Reset. The Command, Status, Request, Temporary and Internal First/Last Flip/Flop registers are cleared and the Mask register is set. The Am9517A will enter the Idle cycle.

Figure 4 lists the address codes for the software commands.

Interface Signals						Operation
A3	A2	A1	A0	IOR	IOW	
1	0	0	0	0	1	Read Status Register
1	0	0	0	1	0	Write Command Register
1	0	0	1	0	1	Illegal
1	0	0	1	1	0	Write Request Register
1	0	1	0	0	1	Illegal
1	0	1	0	1	0	Write Single Mask Register Bit
1	0	1	1	0	1	Illegal
1	0	1	1	1	0	Write Mode Register
1	1	0	0	0	1	Illegal
1	1	0	0	1	0	Clear Byte Pointer Flip/Flop
1	1	0	1	0	1	Read Temporary Register
1	1	0	1	1	0	Master Clear
1	1	1	0	0	1	Illegal
1	1	1	0	1	0	Illegal
1	1	1	1	0	1	Illegal
1	1	1	1	1	0	Write All Mask Register Bits

Figure 4. Register and Function Addressing.

## Am9517A

Channel	Register	Operation	Signals							Internal Flip/Flop	Data Bus DB0-DB7
			CS	IOR	IOW	A3	A2	A1	A0		
0	Base & Current Address	Write	0	1	0	0	0	0	0	0	A0-A7
			0	1	0	0	0	0	0	1	A8-A15
	Current Address	Read	0	0	1	0	0	0	0	0	A0-A7
			0	0	1	0	0	0	0	1	A8-A15
	Base & Current Word Count	Write	0	1	0	0	0	0	1	0	W0-W7
			0	1	0	0	0	0	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	0	0	1	0	W0-W7
			0	0	1	0	0	0	1	1	W8-W15
1	Base & Current Address	Write	0	1	0	0	0	1	0	0	A0-A7
			0	1	0	0	0	1	0	1	A8-A15
	Current Address	Read	0	0	1	0	0	1	0	0	A0-A7
			0	0	1	0	0	1	0	1	A8-A15
	Base & Current Word Count	Write	0	1	0	0	0	1	1	0	W0-W7
			0	1	0	0	0	1	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	0	1	1	0	W0-W7
			0	0	1	0	0	1	1	1	W8-W15
2	Base & Current Address	Write	0	1	0	0	1	0	0	0	A0-A7
			0	1	0	0	1	0	0	1	A8-A15
	Current Address	Read	0	0	1	0	1	0	0	0	A0-A7
			0	0	1	0	1	0	0	1	A8-A15
	Base & Current Word Count	Write	0	1	0	0	1	0	1	0	W0-W7
			0	1	0	0	1	0	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	1	0	1	0	W0-W7
			0	0	1	0	1	0	1	1	W8-W15
3	Base & Current Address	Write	0	1	0	0	1	1	0	0	A0-A7
			0	1	0	0	1	1	0	1	A8-A15
	Current Address	Read	0	0	1	0	1	1	0	0	A0-A7
			0	0	1	0	1	1	0	1	A8-A15
	Base & Current Word Count	Write	0	1	0	0	1	1	1	0	W0-W7
			0	1	0	0	1	1	1	1	W8-W15
	Current Word Count	Read	0	0	1	0	1	1	1	0	W0-W7
			0	0	1	0	1	1	1	1	W8-W15

Figure 5. Word Count and Address Register Command Codes.

## MAXIMUM RATINGS

Storage Temperature  
Ambient Temperature  
VCC with Respect to GND  
All Signal Voltages  
Power Dissipation

The products described herein are not to be used in applications requiring exposure to excessive static charge. It is the user's responsibility to ensure proper handling and storage of the device.

## OPERATING CONDITIONS

## Part Number

Am9517ADC/PC  
Am9517A-1DC/PC  
Am9517A-4DC/PC  
Am9517ADM

## ELECTRICAL CHARACTERISTICS

## Parameter

VOH	
VOL	
VIIH	
VIL	
IIX	
IOZ	
ICC	
CO	
CI	
CIO	

## NOTES:

- Typical values are given for nominal process and nominal power supply.
- Input timing parameters are given for waveforms with rise and fall times of 10 ns. Waveform input signals are otherwise noted.
- Output loading is specified as 10 pF unless otherwise noted.
- The new IOW or IOW-100ns and new IOR or IOR-2TCY-50ns and new TDQ is specified as measured at 25°C for TDQ2 assumption.
- DREQ should be connected from HR to DAC.
- DREQ and DAC Timing diagrams.

**MAXIMUM RATINGS** above which useful life may be impaired

Storage Temperature	-65°C to +150°C
Ambient Temperature Under Bias	-55°C to +125°C
VCC with Respect to VSS	-0.5V to +7.0V
All Signal Voltages with Respect to VSS	-0.5V to +7.0V
Power Dissipation (Package Limitation)	1.5W

The products described by this specification include internal circuitry designed to protect input devices from damaging accumulations of static charge. It is suggested, nevertheless, that conventional precautions be observed during storage, handling and use in order to avoid exposure to excessive voltages.

**OPERATING RANGE**

Part Number	T <sub>A</sub>	VCC
Am9517ADC/PC	0°C to +70°C	5.0V ± 5%
Am9517A-1DC/PC	0°C to +70°C	5.0V ± 5%
Am9517A-4DC/PC	0°C to +70°C	5.0V ± 5%
Am9517ADM	-55°C to +125°C	5.0V ± 10%

**ELECTRICAL CHARACTERISTICS** over operating range (Note 1)

Parameter	Description	Test Conditions	Min	Typ	Max	Unit
V <sub>OH</sub>	Output HIGH Voltage	I <sub>OH</sub> = -200μA	2.4			Volts
		I <sub>OH</sub> = -100μA, (HREQ Only)	3.3			
V <sub>OL</sub>	Output LOW Voltage	I <sub>OL</sub> = 3.2mA			0.4	Volts
V <sub>IH</sub>	Input HIGH Voltage		2.0		VCC+0.5	Volts
V <sub>IL</sub>	Input LOW Voltage		-0.5		0.8	Volts
I <sub>I</sub>	Input Load Current	VSS < V <sub>I</sub> < VCC	-10		+10	μA
I <sub>OZ</sub>	Output Leakage Current	VCC < V <sub>O</sub> < VSS+40	-10		+10	μA
I <sub>CC</sub>	VCC Supply Current	T <sub>A</sub> = +25°C		65	130	mA
		T <sub>A</sub> = 0°C		75	150	
		T <sub>A</sub> = -55°C			175	
C <sub>O</sub>	Output Capacitance	f <sub>c</sub> = 1.0MHz, Inputs = 0V		4	8	pF
C <sub>I</sub>	Input Capacitance			8	15	pF
C <sub>IO</sub>	I/O Capacitance			10	18	pF

**NOTES:**

- Typical values are for T<sub>A</sub> = 25°C, nominal supply voltage and nominal processing parameters.
- Input timing parameters assume transition times of 20ns or less. Waveform measurement points for both input and output signals are 2.0V for High and 0.8V for Low, unless otherwise noted.
- Output loading is 1 Standard TTL gate plus 50pF capacitance unless noted otherwise.
- The new IOW or MEMW pulse width for normal write will be TCY-100ns and for extended write will be 2TCY-100ns. The net IOR or MEMR pulse width for normal read will be 2TCY-50ns and for compressed read will be TCY-50ns.
- TDQ is specified for two different output HIGH levels. TDQ1 is measured at 2.0V. TDQ2 is measured at 3.3V. The value for TDQ2 assumes an external 3.3kΩ pull-up resistor connected from HREQ to VCC.
- DREQ should be held active until DACK is returned.
- DREQ and DACK signals may be active high or active low. Timing diagrams assume the active high mode.
- Output loading on the data bus is 1 Standard TTL gate plus 15pF for the minimum value and 1 Standard TTL gate plus 100pF for the maximum value.
- Successive read and/or write operations by the external processor to program or examine the controller must be timed to allow at least 600ns for the Am9517A or Am9517A-1 and at least 450ns for the Am9517A-4 as recovery time between active read or write pulses.
- Parameters are listed in alphabetical order.
- Pin 5 is an input that should always be at a logic high level. An internal pull-up resistor will establish a logic high when the pin is left floating. Alternatively, pin 5 may be tied to VCC.
- Signals HEAD and WHITE refer to IOR and MEMW respectively for peripheral-to-memory DMA operations and to MEMR and IOW respectively for memory-to-peripheral DMA operations.
- If N wait states are added during the write-to-memory half of a memory-to-memory transfer, this parameter will increase by N (TCY).

## Am9517A

## SWITCHING CHARACTERISTICS

ACTIVE CYCLE (Notes 2, 3, 10, 11 and 12)

Parameter	Description	Am9517A		Am9517A-1		Am9517A-4		Unit
		Min	Max	Min	Max	Min	Max	
TAEL	AEN HIGH from CLK LOW (S1) Delay Time		300		300		225	ns
TAET	AEN LOW from CLK HIGH (S1) Delay Time		200		200		150	ns
TAFAB	ADR Active to Float Delay from CLK HIGH		150		150		120	ns
TAFC	READ or WRITE Float from CLK HIGH		150		150		120	ns
TAFDB	DB Active to Float Delay from CLK HIGH		250		250		190	ns
TAHR	ADR from READ HIGH Hold Time	TCY 100		TCY 100		TCY 100		ns
TAHS	DB from ADSTB LOW Hold Time	50		50		40		ns
TAHW	ADR from WRITE HIGH Hold Time	TCY 50		TCY 50		TCY 50		ns
TAK	DACK Valid from CLK LOW Delay Time		280		280		220	ns
	EOP HIGH from CLK HIGH Delay Time		250		250		190	ns
	EOP LOW to CLK HIGH Delay Time		250		250		190	ns
TASM	ADR Stable from CLK HIGH		250		250		190	ns
TASS	DB to ADSTB LOW Setup Time	100		100		100		ns
TCH	Clock High Time (Transitions $\leq$ 10ns)	120		120		100		ns
TCL	Clock Low Time (Transitions $\leq$ 10ns)	150		150		110		ns
TCY	CLK Cycle Time	320		320		250		ns
TDCL	CLK HIGH to READ or WRITE LOW Delay (Note 4)		270		270		200	ns
TDCTR	READ HIGH from CLK HIGH (S4) Delay Time (Note 4)		270		270		210	ns
TDCTW	WRITE HIGH from CLK HIGH (S4) Delay Time (Note 4)		200		200		150	ns
TDQ1	HREQ Valid from CLK HIGH Delay Time (Note 5)		160		160		120	ns
TDQ2			250		250		190	ns
TEPS	EOP LOW from CLK LOW Setup Time	60		60		45		ns
TEPW	EOP Pulse Width	300		300		225		ns
TFAAB	ADR Float to Active Delay from CLK HIGH		250		250		190	ns
TFAC	READ or WRITE Active from CLK HIGH		200		200		150	ns
TFADB	DB Float to Active Delay from CLK HIGH		300		300		225	ns
THS	HACK valid to CLK HIGH Setup Time	100		100		75		ns
TIDH	Input Data from MEMR HIGH Hold Time	0		0		0		ns
TIDS	Input Data to MEMR HIGH Setup Time	250		250		190		ns
TODH	Output Data from MEMW HIGH Hold Time	200		200		200		ns
TODV	Output Data Valid to MEMW HIGH (Note 13)	200		200		125		ns
TQS	DREQ to CLK LOW (S1, S4) Setup Time	120		120		90		ns
TRH	CLK to READY LOW Hold Time	20		20		20		ns
TRS	READY to CLK LOW Setup Time	100		100		60		ns
TSTL	ADSTB HIGH from CLK HIGH Delay Time		200		200		150	ns
TSTT	ADSTB LOW from CLK HIGH Delay Time		140		140		110	ns

SWITCHING  
PROGRAM C  
(Notes 2, 3, 1)

## Parameter

TAR	
TAW	
TCW	
TDW	
TRA	
TRDE	
TDRF	
TRSTD	
TRSTS	
TRSTW	
TRW	
TWA	
TWC	
TWD	
TWWS	
TAD	

## SWITCHING CHARACTERISTICS (Cont.)

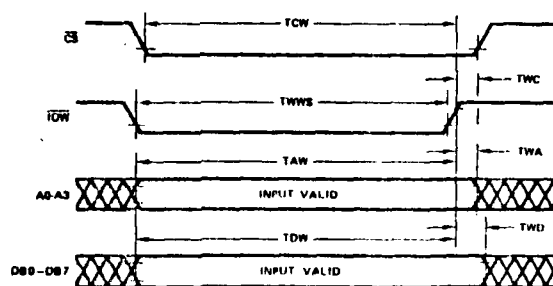
PROGRAM CONDITION (IDLE CYCLE)

(Notes 2, 3, 10, 11 and 12)

Max	Unit
225	ns
150	ns
120	ns
120	ns
190	ns
	ns
	ns
	ns
220	ns
190	ns
190	ns
190	ns
	ns
	ns
	ns
	ns
200	ns
210	ns
150	ns
120	ns
	ns
	ns
190	ns
150	ns
225	ns
	ns
	ns
	ns
	ns
	ns
	ns
	ns
150	ns
110	ns

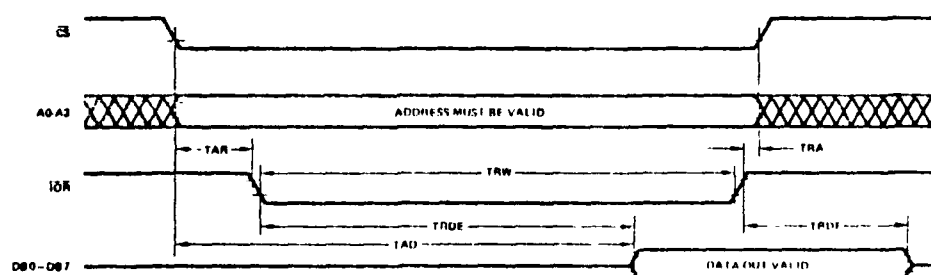
Parameter	Description	Am9517A		Am9517A-1		Am9517A-4		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
TAR	ADR Valid or $\overline{CS}$ LOW to $\overline{RD}$ LOW	50		50		50		ns
TAW	ADR Valid to $\overline{WR}$ HIGH Setup Time	200		200		150		ns
TCW	$\overline{CS}$ LOW to $\overline{WR}$ HIGH Setup Time	200		200		150		ns
TDW	Data Valid to $\overline{WR}$ HIGH Setup Time	200		200		150		ns
TRA	ADR or $\overline{CS}$ Hold from $\overline{RD}$ HIGH	0		0		0		ns
TRDE	Data Access from $\overline{RD}$ LOW (Note 8)		300		200		200	ns
TDRF	DB Float Delay from $\overline{RD}$ HIGH	20	150	20	100	20	100	ns
TRSTD	Power Supply HIGH to RESET LOW Setup Time	500		500		500		$\mu$ s
TRSTS	RESET to First $\overline{IOWR}$	2		2		2		TCY
TRSTW	RESET Pulse Width	300		300		300		ns
TRW	$\overline{RD}$ Width	300		300		250		ns
TWA	ADR from $\overline{WR}$ HIGH Hold Time	20		20		20		ns
TWC	$\overline{CS}$ HIGH from $\overline{WR}$ HIGH Hold Time	20		20		20		ns
TWD	Data from $\overline{WR}$ HIGH Hold Time	30		30		30		ns
TWWS	Write Width	200		200		200		ns
TAD	Data Access from ADR Valid, $\overline{CS}$ LOW		350		300		300	ns

## SWITCHING WAVEFORMS



Timing Diagram 1. Program Condition Write Timing (Note 9).

MOS-036

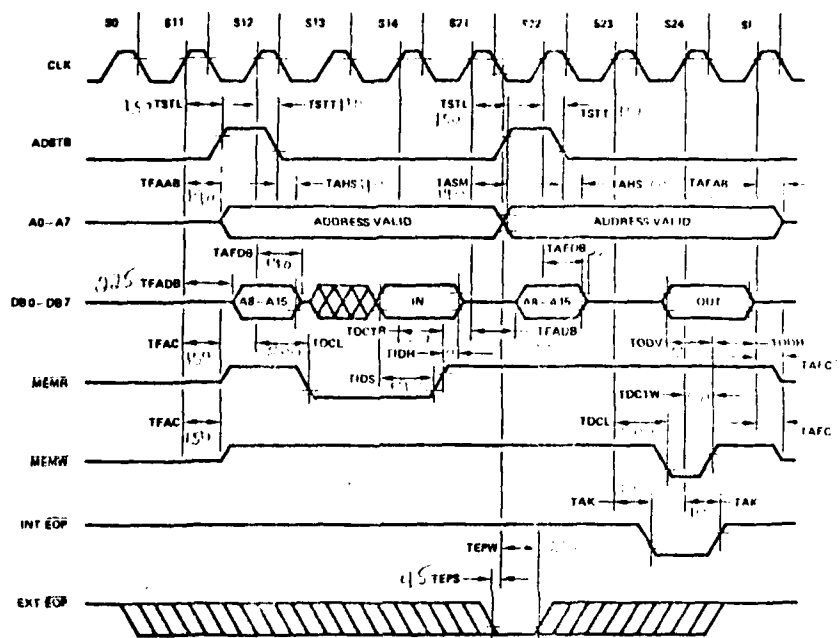


Timing Diagram 2. Program Condition Read Cycle (Note 9).

MOS-037

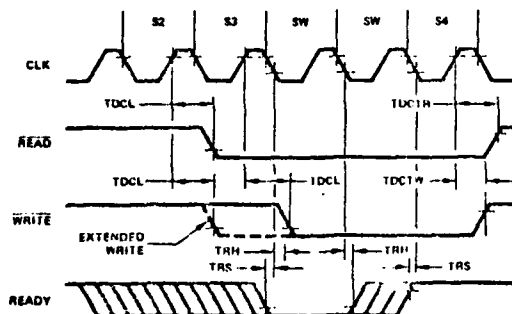


### SWITCHING WAVEFORMS (Cont.)



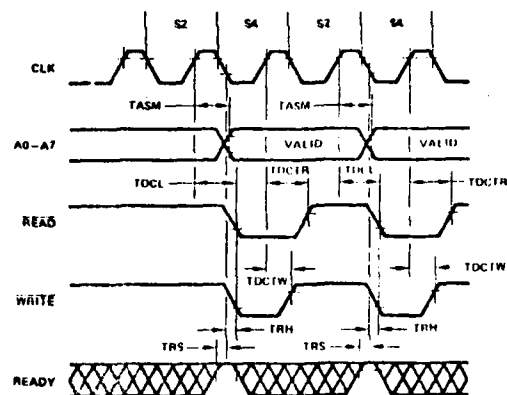
**Timing Diagram 4. Memory-to-Memory.**

WJS 039



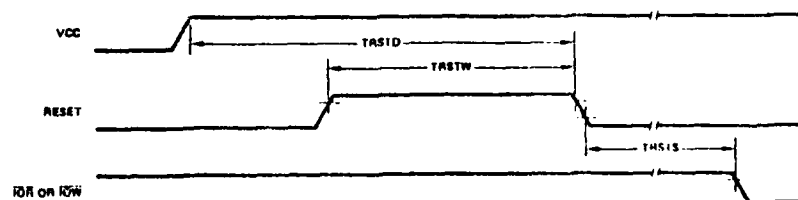
### Timing Diagram 5. Ready Timing.

**MOS-040**



### Timing Diagram 6. Compressed Timing.

MOS 041



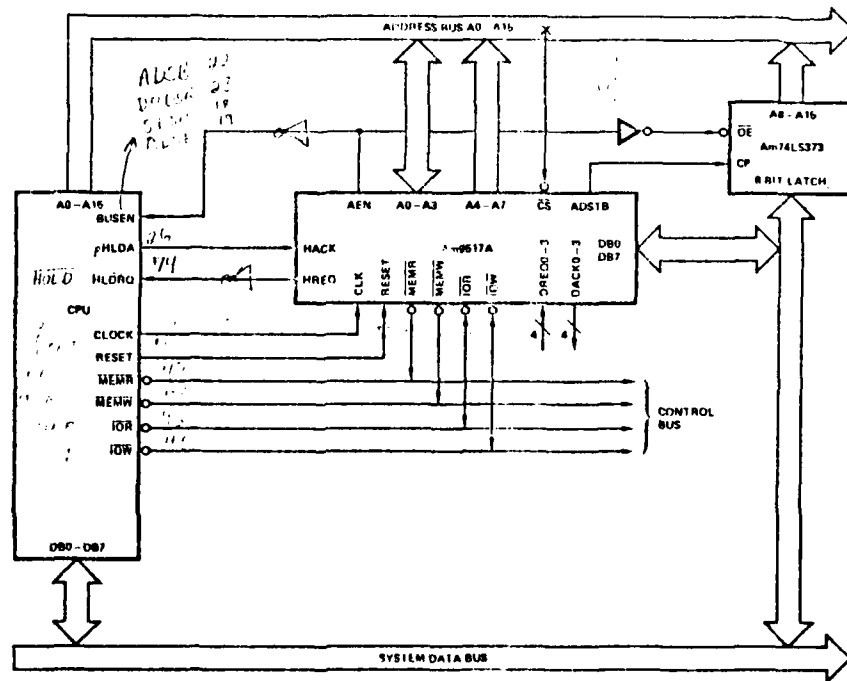
### Timing Diagram 7. Reset Timing.

MOS 042

## APPLICATION INFORMATION

Figure 6 shows a convenient method for configuring a DMA system with the Am9517A Controller and a microprocessor system. The Multimode DMA Controller issues a Hold Request to the processor whenever there is at least one valid DMA Request from a peripheral device. When the processor replies with a Hold Acknowledge signal, the Am9517A takes control of the Address Bus, the Data Bus and the Control Bus. The address for the first transfer operation comes out in two bytes - the least significant eight bits on the eight Address outputs and the most

significant eight bits on the Data Bus. The contents of the Data Bus are then latched into the Am74LS373 register to complete the full 16 bits of the Address Bus. The Am74LS373 is a high speed, low power, 8-bit, 3-state register in a 20-pin package. After the initial transfer takes place, the register is updated only after a carry or borrow is generated in the least significant address byte. Four DMA channels are provided when one Am9517A is used.



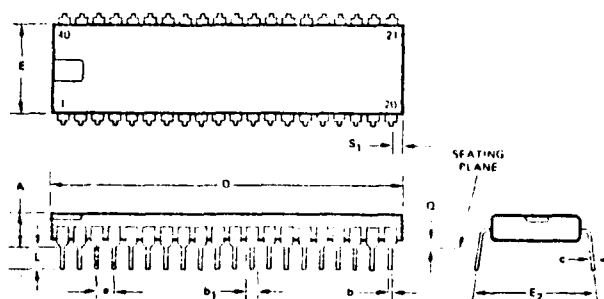
MOS-043

Figure 6. Basic DMA Configuration.

the contents of the Data 3 register to complete Am74LS373 is a high in a 20-pin package. Register is updated only the least significant address provided when one

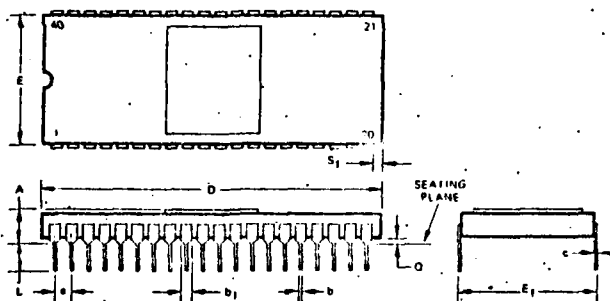
# PHYSICAL DIMENSIONS Dual-In-Line

## 40-Pin Plastic



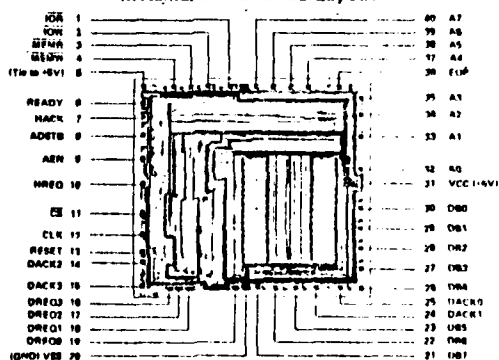
Reference Symbol	Inches	
	Min.	Max.
A	.150	.200
b	.015	.020
b <sub>1</sub>	.055	.065
c	.009	.011
D	2.050	2.080
E	.530	.590
E <sub>1</sub>	.585	.700
e	.090	.110
L	.125	.160
Q	.015	.060
S <sub>1</sub>	.040	.070

## 40-Pin Hermetic



Reference Symbol	Inches	
	Min.	Max.
A	.100	.200
b	.015	.022
b <sub>1</sub>	.030	.060
c	.008	.013
D	1.960	2.040
E	.550	.610
E <sub>1</sub>	.590	.620
e	.090	.110
L	.120	.160
Q	.020	.060
S <sub>1</sub>	.005	

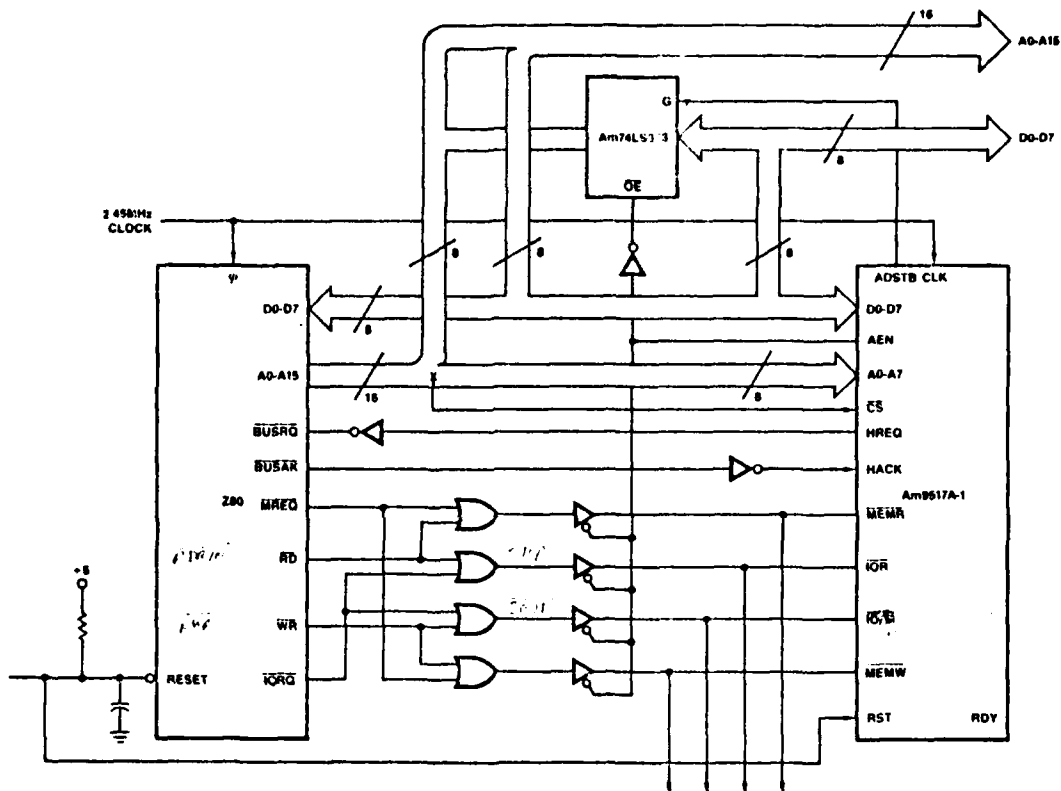
## Metallization and Pad Layout



DIE SIZE  
0.198" X 0.210"

# Am9517A-1 to Z80 Interface

CIRCUIT DIAGRAM:



## DESCRIPTION OF INTERFACE:

In this example, the high order A<sub>8</sub>-A<sub>15</sub> memory address is latched by an 8-bit Am74LS373 latch, while the IOR, IO/M, MEMR and MEMW signals are decoded from appropriate Z80 outputs. Note that a 4MHz Am9517A-4 could be used with a

4MHz Z80A if an additional circuit consisting of two D flip-flops are used to insert a wait state to the Z80A during I/O read and write operations.

## DESCRIPTION

The Am9517A (6800 microprocessor) holds the high order A<sub>8</sub>-A<sub>15</sub> memory address. A D-type flip-flop (HREQ) signal. The HREQ and Am9517A's clock

## PRELIMINARY INFORMATION

### DISTINCTIVE CHARACTERISTICS

- Five independent 16-bit counters
- High speed counting rates
- Up/down and binary/BCD counting
- Internal oscillator frequency source
- Tapped frequency scaler
- Programmable frequency output
- 8-bit or 16-bit bus interface
- Time-of-day option
- Alarm comparators on counters 1 and 2
- Complex duty cycle outputs
- One-shot or continuous outputs
- Programmable count/gate source selection
- Programmable input and output polarities
- Programmable gating functions
- Retriggering capability
- +5 volt power supply
- Standard 40 pin package
- 100% MIL-STD-883 reliability assurance testing

### GENERAL DESCRIPTION

The Am9513 System Timing Controller is an LSI circuit designed to service many types of counting, sequencing and timing applications. It provides the capability for programmable frequency synthesis, high resolution programmable duty cycle waveforms, retriggerable digital one-shots, time-of-day clocking, coincidence alarms, complex pulse generation, high resolution baud rate generation, frequency shift keying, stop-watching timing, event count accumulation, waveform analysis and many more. A variety of programmable operating modes and control features allow the Am9513 to be personalized for particular applications as well as dynamically reconfigured under program control.

The STC includes five general-purpose 16-bit counters. A variety of internal frequency sources and external pins may be selected as inputs for individual counters with software selectable active-high or active-low input polarity. Both hardware and software gating of each counter is available. Three-state outputs for each counter provide pulses or levels and can be active-high or active-low. The counters can be programmed to count up or down in either binary or BCD. The host processor may read an accumulated count at any time without disturbing the counting process. Any of the counters may be internally concatenated to form any effective counter length up to 80 bits.

### GENERAL BLOCK DIAGRAM

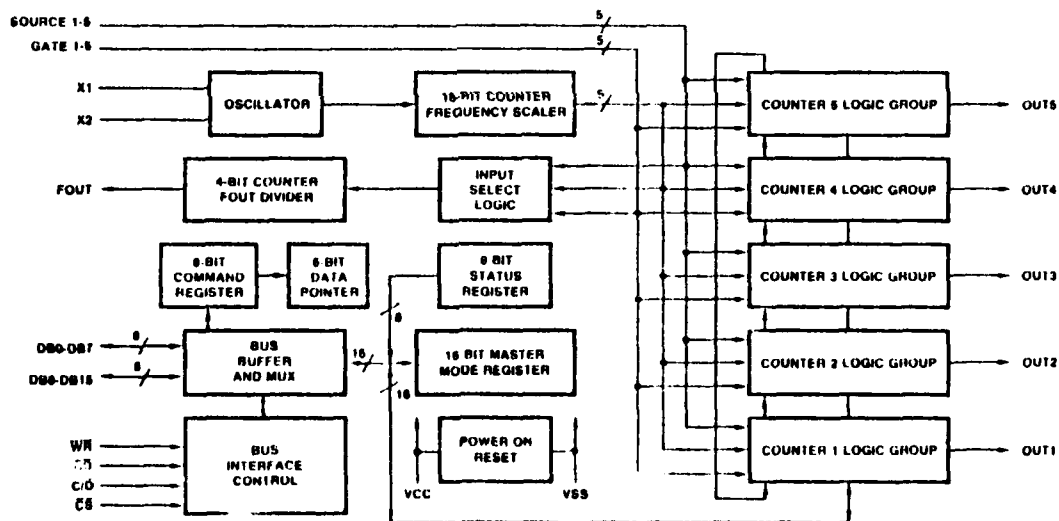


Figure 1.

MOS 169

### ORDERING INFORMATION

Package Type	Temperature Range	Counting Frequency	
		7MHz	
Molded	0°C ≤ T <sub>A</sub> ≤ +70°C	AM9513PC	
Hermetic Side-Brazed Ceramic		AM9513DC	
Hermetic Cerdip		AM9513CC	
Hermetic Side Brazed Ceramic	-55°C ≤ T <sub>A</sub> ≤ +125°C	AM9513DM	

## FUNCTIONAL DESCRIPTION

The Am9513 block diagrams (Figures 1 and 2) indicate the interface signals and the basic flow of information. Internal control lines and the internal data bus have been omitted. The control and data registers are all connected to a common internal 16-bit bus. The external bus may be 8 or 16 bits wide; in the 8-bit mode the internal 16-bit information is multiplexed to the low order data bus pins DB0 through DB7.

An internal oscillator provides a convenient source of frequencies for use as counter inputs. Its oscillating frequency is controlled by an external reactive network such as a crystal. The oscillator output is divided by the Frequency Scaler to provide several sub-frequencies. One of the scaled frequencies (or one of ten input signals) may be selected as an input to the FOUT interface pin and then comes out of the chip at the FOUT interface pin.

The STC is addressed by the external system as two locations: a control port and a data port. The control port provides direct access to the Status and Command registers, as well as allowing the user to update the Data Pointer register. The data port is used to communicate with all other addressable internal locations. The Data Pointer controls the data port addressing.

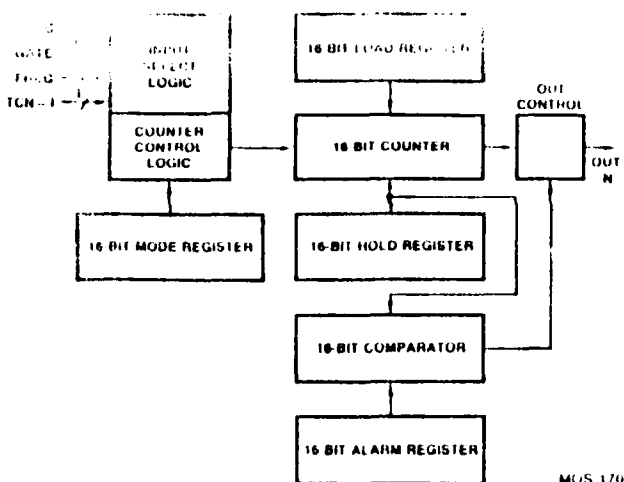
Among the registers accessible through the data port are the Master Mode register and five Counter Mode registers, one for each counter. The Master Mode register controls the programmable options that are not controlled by the Counter Mode registers.

Each of the five general purpose counters is 16 bits long and is independently controlled by its Counter Mode register. Through this register, a user can software select one of 16 sources as the counter input, a variety of gating and repetition modes, up or down counting in binary or BCD and active-high or active-low input and output polarities.

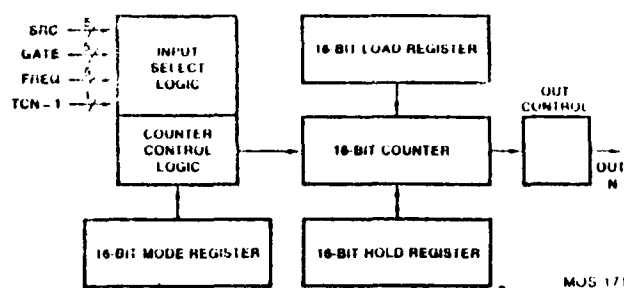
Associated with each counter is a Load register and a Hold register, both accessible through the data port. The Load register is used to automatically reload the counter to any predefined value, thus controlling its effective period. The Hold register is used to save count values without disturbing the count process, permitting the host processor to read intermediate counts. In addition, the Hold register may be used as a second Load register to generate a number of complex output waveforms.

All five counters have the same basic control logic and control registers. Counters 1 and 2 have additional alarm registers and comparators associated with them, plus the extra logic necessary for operating in a 24-hour time-of-day mode. For real-time operation the time-of-day logic will accept 50Hz, 60Hz or 100Hz input frequencies.

Each general counter has a single dedicated output pin. It may be turned off when the output is not of interest or may be configured in a variety of ways to drive interrupt controllers, Darlington buffers, bus drivers, etc. The counter inputs, on the other hand, are specifically not dedicated to any given interface line. Considerable versatility is available for configuring both the input and the gating of individual counters. This not only permits dynamic reassignment of inputs under software control, but also allows multiple counters to use a single input, and allows a single gate pin to control more than one counter.



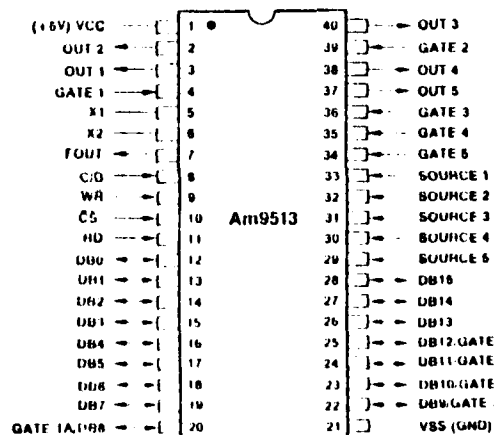
Counter Logic Groups 1 and 2.



Counter Logic Groups 3, 4, and 5.

Figure 2.

## CONNECTION DIAGRAM



Top View

Pin 1 is marked for orientation

MOS 172

Figure 3.

## INTERFACE SIGNAL DESCRIPTION

**VCC:** +5 volt power supply

**VSS:** Ground

### X1,X2 (Crystal, Inputs)

X1 and X2 are the connections for an external crystal that determines the frequency of the internal oscillator. An RC or LC network may also be used instead of a crystal. For driving from an external frequency source, X1 should be left open and X2 should be driven with a TTL-level square wave.

### FOUT (Frequency Out, Output)

The FOUT output is derived from a 4-bit counter that may be programmed to divide its input by any integer value from 1 to 16, inclusive. The input to the counter is selected from any of 15 sources, including the scaled internal frequencies. FOUT may be gated on and off under software control. Following power-up or reset, FOUT provides a frequency that is 1/16 that of the internal oscillator.

### GATE1-GATE5 (Gate, Inputs)

The Gate inputs provide hardware control of the counting operations of individual counters by determining when counting may proceed. The same input may control up to three counters. Gates may also be selected as count sources for any of the counters or for the FOUT divider. The active polarity for a selected Gate input is programmable at each counter. Schmitt-trigger circuitry on the GATE inputs allows slow transition times to be used.

### SRC1-SRC5 (Source, Inputs)

The Source inputs provide external signals that may be counted by any of the counters. Any Source line may be routed to any or all of the counters and the FOUT divider. The active polarity for a selected Source input is programmed at each counter. Any source waveform duty cycle will be accepted as long as the minimum pulse width is at least half the period of the maximum specified counting frequency for the part. Schmitt-trigger circuitry on the SRC inputs allows slow transition times to be used.

### OUT1-OUT5 (Counter Outputs)

Each of the five counters has a dedicated output pin. Depending on the output configuration, the OUT signal may be a pulse, a square wave, or a complex duty cycle waveform. For counters 1 and 2, the OUT signal may also indicate the status of comparator circuits. Output polarities may be individually programmed.

### DB0-DB7, DB8-DB15 (Data Bus, Input/Output)

### GATE1A-GATE5A (Auxiliary Gates, Input)

The Data Bus lines are used to communicate with the external system. After power-up or reset, the data bus will be configured for 8-bit width. It may be reconfigured for 16-bit width by changing a control bit in the Master Mode register. Figure 4 summarizes all data bus transfers.

When operating in the 8-bit data bus environment, DB13, DB14, and DB15 may optionally be used as additional  $\overline{CS}$  lines. If Figure 3 is unused they should be held high. When pulled low, a  $\overline{Gate}/\overline{A}$  signal will disable the action of the gate input controlling counter N. DB13, DB14 and DB15 should be tied high for an 8-bit data bus width.

### $\overline{CS}$ (Chip Select, Input)

The active-low Chip Select input enables Read and Write operations on the data bus. See Figure 4.

### $\overline{RD}$ (Read, Input)

The active-low Read signal is conditioned by Chip Select and indicates that internal information is to be transferred to the data bus.  $\overline{WR}$  and  $\overline{RD}$  should be mutually exclusive.

### $\overline{WR}$ (Write, Input)

The active-low Write signal is conditioned by Chip Select and indicates that data bus information is to be transferred to an internal location.  $\overline{WR}$  and  $\overline{RD}$  should be mutually exclusive.

### $\overline{C/D}$ (Control/Data, Input)

The Control/Data signal selects source and destination locations for read and write operations on the data bus. Control Write operations load the Command register and the Data Pointer. Control Read operations output the Status register. Data Read and Data Write transfers communicate with all other internal registers.

Signal Configuration				Data Bus Operation
$\overline{CS}$	$\overline{C/D}$	$\overline{RD}$	$\overline{WR}$	
0	0	0	1	Transfer contents of register addressed by Data Pointer to the data bus.
0	0	1	0	Transfer contents of data bus to data register addressed by Data Pointer.
0	1	0	1	Transfer contents of Status register to data bus.
0	1	1	0	Transfer contents of data bus into Command register.
X	X	1	1	No transfer.
1	X	X	X	No transfer.
X	X	0	0	Illegal Condition.

Figure 4. Data Bus Transfers.

## CONTROL PORT COMMANDS

### Command Register

The 8-bit write-only Command register is loaded by writing into the control port as shown in Figure 4. With a 16-bit data bus, the low-order 8 bits are loaded into the register, the high-order byte should be FF (hex).

The Command register provides direct control over each of the five general counters and controls access through the data port by allowing the user to update the Data Pointer register. A summary of all commands appears in Figure 5. Six of the command types are used for direct software control of the counting process. Each of these six commands contains a five bit S field. In a linear-select fashion, each bit in the S field corresponds to one of the five general counters (S1 = Counter 1, S2 = Counter 2, etc.). When an S bit is a one, the specified operation is performed on the counter so designated; when an S bit is a zero, no operation occurs for the corresponding counter.

A counter must be armed by one of the ARM commands before counting can commence. Once armed, the counting process may be further enabled or disabled using the hardware gating facilities. The ARM and DISARM commands permit software gating of the count process, in some modes.

The LOAD command causes the counter to be reloaded with the value in either the associated Load register or the associated Hold register. It will often be used as a software retrigger, or as counter initialization prior to active hardware gating.

The DISARM command disables further counting independent of any hardware gating. A disarmed counter may be reloaded using the LOAD command, may be incremented or decremented using the STEP command and may be read using the SAVE command. A count process may be resumed using an ARM command.

The SAVE command transfers the contents of a counter to its associated Hold register. The transfer takes place without interfering with any counting that may be under way. This command will overwrite any previous Hold register contents. The SAVE command is designed to allow an accumulated count to be preserved so that it can be read by the host CPU at some later time.

Two other classes of the basic commands exist: ARM AND ARM AND DISARM AND SAVE. Any combination of these three commands are provided to step an integration counter by one count, set and clear an output toggle, issue a software reset, clear and set special bits in the Master Mode register, load the Data Pointer register.

### Data Pointer Register

The 8-bit Data Pointer register is loaded by issuing the appropriate command through the control port to the Command register. As shown in Figure 6, the Data Pointer register consists of a Byte Pointer, an Element Pointer, and a Group Pointer. The content of the Data Pointer is used as an address to point to an internal register. When a register is addressed by the Data Pointer, it may be accessed through the data port.

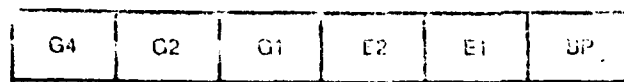
The Byte Pointer bit in the Data Pointer register indicates which byte of a 16-bit register is to be transferred on the next access through the data port. Whenever the Data Pointer is loaded, the Byte Pointer bit is set to one, indicating a least-significant byte is expected. The Byte Pointer toggles following each 8-bit data transfer with an 8-bit data bus (MM13 = 0), or it always remains set with the 16-bit data bus option (MM13 = 1). Although the contents of the Element and Group Pointer in the Data Pointer register cannot be read by the host processor, the Byte Pointer is available as a bit in the Status register.

To permit the host processor to rapidly access the various internal registers, automatic sequencing of the Data Pointer is provided. Sequencing is enabled by clearing Master Mode bit 14 (MM14) to 0. As shown in Figure 7, several types of sequencing are available depending on the data bus width being used and the initial Data Pointer value entered by command.

When E1 = 0 or L2 = 0 and G4, G2, G1 point to a Counter Group, the Data Pointer will proceed through the Element cycle. The Element field will automatically sequence through three values: 00, 01 and 10 starting with the value entered. When the transition from 10 to 00 occurs, the Group field will also be incremented by one. Note that the Element field in this case does not sequence to a value of 11. The Group field circulates only within the five Counter Group codes.

Command Code								Command Description
C7	C6	C5	C4	C3	C2	C1	C0	
0	0	0	E2	E1	G4	G2	G1	Load Data Pointer register with contents of E and G fields (G ≠ 000, G ≠ 110)
0	0	1	S5	S4	S3	S2	S1	Arm counting for all selected counters
0	1	0	S5	S4	S3	S2	S1	Load contents of specified source into all selected counters
0	1	1	S5	S4	S3	S2	S1	Load and Arm all selected counters
1	0	0	S5	S4	S3	S2	S1	Disarm and Save all selected counters
1	0	1	S5	S4	S3	S2	S1	Save all selected counters in hold register
1	1	0	S5	S4	S3	S2	S1	Disarm all selected counters
1	1	1	0	1	N4	N2	N1	Set output bit N (001 ≤ N ≤ 101)
1	1	1	0	0	N4	N2	N1	Clear output bit N (001 ≤ N ≤ 101)
1	1	1	1	0	N4	N2	N1	Step counter N (001 ≤ N ≤ 101)
1	1	1	0	1	0	0	0	Set MM14 (Disable Data Pointer Sequencing)
1	1	1	0	1	1	1	0	Set MM12 (code 011001)
1	1	1	0	1	1	1	1	Set MM13 (Enter 16 bit bus mode)
1	1	1	0	0	0	0	0	Clear MM14 (Enable Data Pointer Sequencing)
1	1	1	0	0	1	1	0	Clear MM12 (code 01001)
1	1	1	0	0	1	1	1	Clear MM13 (Enter 8 bit bus mode)
1	1	1	1	1	1	1	1	Master reset

Figure 5. Am9513 Command Summary.



#### Byte Pointer

- 1 = Least significant Byte Transferred next
- 0 = Most significant Byte Transferred next

#### Group Pointer

- 000 = Illegal
- 001 = Counter Group 1
- 010 = Counter Group 2
- 011 = Counter Group 3
- 100 = Counter Group 4
- 101 = Counter Group 5
- 110 = Illegal
- 111 = Control Group

#### Element Pointer

- 00 = Mode Register
- 01 = Load Register
- 10 = Hold Register
- 11 = Hold Register/Hold Cycle Increment

- 00 = Alarm Register 1
- 01 = Alarm Register 2
- 10 = Master Mode Register
- 11 = Status Register/No Increment

Control Cycle  
Increment

Figure 6. Data Pointer Counter.

MOS 173

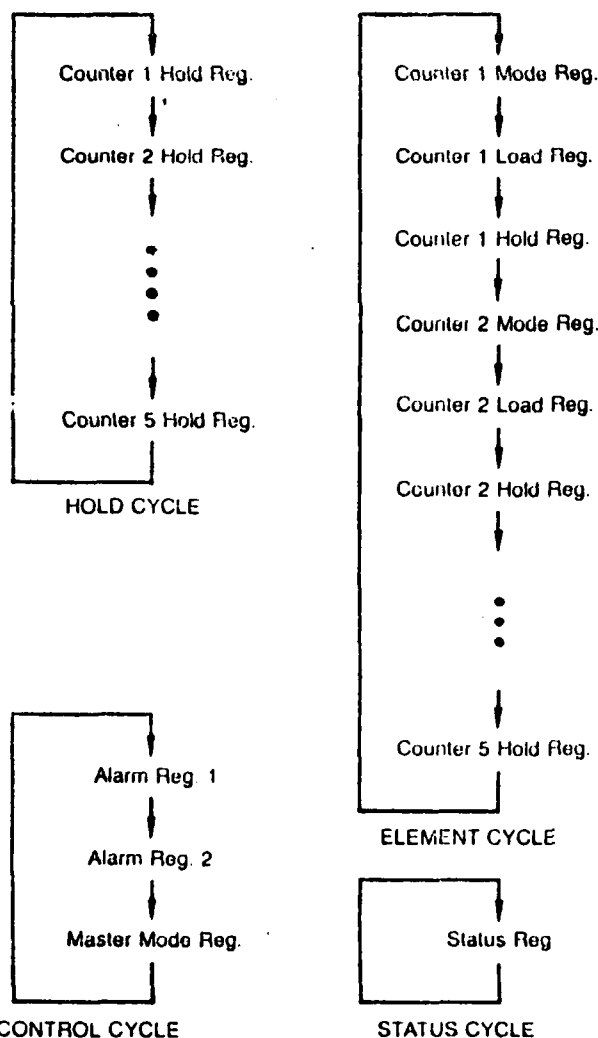


Figure 7. Data Pointer Sequencing.

MOS-174

If E1 = 1 and E2 = 1, then only the Group field is sequenced. This is the Hold cycle. It allows the Hold registers to be sequentially accessed while bypassing the Mode and Load registers. The third type of sequencing is the Control cycle. If G4, G2, G1 = 111 and E2, E1 ≠ 11, the Element Pointer will be incremented through the values 00, 01 and 10, with no change to the Group Pointer.

When G4, G2, G1 = 111 and E2, E1 = 11, no incrementing takes place and only the Status register will be available through the data port. Note that the Status register can also always be read directly through the Control port.

For all of these auto-sequence modes, if an 8-bit data bus is used, the Byte pointer will toggle after every data transfer to allow the least and most significant bytes to be transferred before the Element or Group Fields are incremented.

#### Status Register

The 8-bit read-only Status register indicates the state of the Byte Pointer bit in the Data Pointer register and the state of the OUT signal for each of the general counters. See Figure 8. The OUT signals reported are those internal to the chip after the polarity-select logic and just before the three-state interface buffer circuitry. The Status register is normally accessed by reading the control port (see Figure 4) but may also be read via the data port as part of the Control Group.

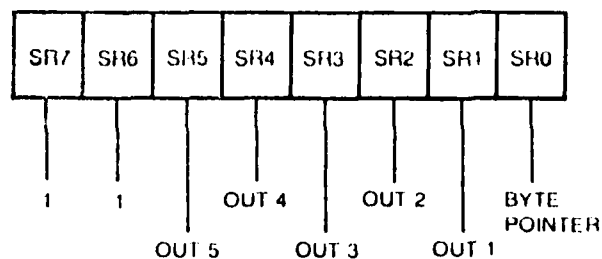


Figure 8. Status Register Bit Assignments.

MOS 175

## DATA PORT REGISTER

### Counter Logic Groups

As shown in Figure 2, each of the five Counter Logic Groups consists of a 16-bit general counter with associated control and output logic, a 16-bit Load register, a 16-bit Hold register and a 16-bit Mode register. In addition, Counter Groups 1 and 2 also include 16-bit Comparators and 16-bit Alarm registers. The comparator/alarm functions are controlled by the Master Mode register. The operation of the Counter Mode registers, then, is the same for all five counters. The host CPU has both read and write access to all registers in the Counter Logic Groups through the data port. The counter itself is never directly accessed.

The 16-bit read/write Load register is used to control the effective period of the general counter. Any 16-bit value may be written into the Load register. That value can then be transferred into the counter each time that Terminal Count (TC) occurs. "Terminal Count" is defined as that period of time when the counter contents would have been zero if an external value had not been transferred into the counter. Thus the terminal count frequency can be the input frequency divided by the value in the Load register. In all operating modes the contents of either Load or Hold will be transferred into the counter when TC occurs. In cases where values are being accumulated in the counter, the Load register action can be transparent by filling the Load register with all zeros.

The 16-bit read/write Hold register is dual purpose. It can be used in the same way as the Load register, thus offering an alternate source for modulo definition for the counter. The Hold register may also be used to store accumulated counter values for later

transfer to the Load register. This allows the counter to be placed in a hold state as the next operation proceeds. Transfer of counter contents into the Hold register is accomplished by the hardware interface in some operating modes or by the software SAVE command at any time.

The 16-bit read/write Counter Mode register controls the gating, counting, output and source select functions within each Counter Logic Group. Figure 9 shows the bit assignments for the Counter Mode registers. Generally each counter is independently configured by its Counter Mode register and does not depend on configuration information outside its Counter Logic Group.

Counter mode bits CM0 through CM2 specify the output control configuration. The OUT pin may be off and in a high impedance state, or it may be off with a low impedance to ground. The six remaining combinations are split into active-high and active-low versions of the three basic output waveforms.

One output form available is called Terminal Count (TC) and represents the period in time that the counter reaches an equivalent value of zero. Figure 10 shows a Terminal Count pulse and an example context that generated it. The TC width is determined by the period of the counting source. Regardless of any gating input, the terminal count will go active for only one clock cycle. Figure 10 assumes active-high source polarity, counter armed, counter decrementing and an external reload value of K.

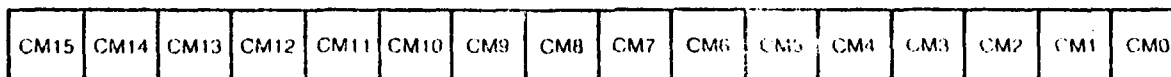
The counter will always be loaded from an external location when TC occurs; the user can choose the source location and the value. If a non-zero value is picked, the counter will never really attain a zero state and TC will indicate the counter state that would have been zero had no parallel transfer occurred.

### Count Source Selection

0XXXX = Count on Rising Edge  
1XXXX = Count on Falling Edge  
X0000 = TCN 1  
X0001 = SRC 1  
X0010 = SRC 2  
X0011 = SRC 3  
X0100 = SRC 4  
X0101 = SRC 5  
X0110 = GATE 1  
X0111 = GATE 2  
X1000 = GATE 3  
X1001 = GATE 4  
X1010 = GATE 5  
X1011 = F1  
X1100 = F2  
X1101 = F3  
X1110 = F4  
X1111 = F5

### Count Control

0XXXX Disable Special Gate  
1XXXX Enable Special Gate  
X0XXX Reload from Load  
X1XXX Reload from Load or Hold  
XX0XX Count Once  
XX1XX Count Repetitively  
XX00X Binary Count  
XX10X BCD Count  
XXX00 Count Down  
XXX01 Count Up



### Gating Control

000 No Gating  
001 Active High Level TCN 1  
010 Active High Level GATE N + 1  
011 Active High Level GATE N - 1  
100 Active High Level GATE N  
101 Active Low Level GATE N  
110 Active High Edge GATE N  
111 Active Low Edge GATE N

### Output Control

000 Inactive, Output Low  
001 Active High Terminal Count Pulse  
010 Active High Toggle, Delayed  
~~011 Active High Toggle, Immediate~~  
100 Inactive, Output High Impedance  
101 Active Low Terminal Count Pulse  
~~110 Active Low Toggle, Delayed~~  
~~111 Active Low Toggle, Immediate~~

Figure 9. Counter Mode Register Bit Assignments.

MOS 178

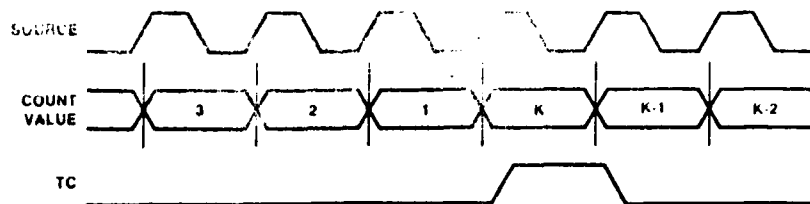


Figure 10. Terminal Count Waveform.

MOS 177

Another output form uses TC to toggle a flip flop to generate an output level instead of a pulse. Two variations of the toggle waveforms are available, as shown in Figure 11. The one labeled "Delayed" uses only the TC pulse to change the toggle. Since TC does not occur until a full count elapses following the loading of the counter, the first transition of the toggle is delayed from the moment of arming. On the other hand, the waveform labeled "Immediate" also uses the TC pulse as the toggling source but adds a toggle transition on the first count following the arming. After the initial transition, both Delayed and Immediate waveforms are the same; for the same output polarity they will be  $180^\circ$  out of phase. The trailing edge of TC triggers the toggle and the toggle output is  $1/2$  the frequency of TC.

Counter Mode bits CM8 through CM12 specify the source used as input to the counter and the active edge that is counted. Bit CM12 controls the polarity for all the sources, logic zero counts rising edges and logic one counts falling edges. Bits CM8 through CM11 select one of sixteen counting sources to route to the counter input. Five of the available inputs are internal frequencies derived from the internal oscillator (see Figure 15 for frequency assignments). Ten of the available inputs are interface pins; five are labeled SRC and five are labeled GATE. The sixteenth available input is the TC signal from the adjacent lower-numbered counter (The Counter 5 TC wraps around to the Counter 1 input). This option allows internal concatenation that permits very long counts to be accumulated. When TCN 1 is the source, the count ripples between the connected counters.

Counter Mode bits CM3 through CM7 specify the various options available for direct control of the counting process. CM3 and CM4 operate independently of the others and control up/down and BCD/binary counting. They may be combined freely with other control bits to form many types of counting configurations. The other three bits interact in complex ways. Bit CM5 controls the repetition of the count process. When CM5 = 1, counting will proceed in the specified mode until the counter is disarmed or the mode is changed. When CM5 = 0 the count process will proceed only until one full cycle of operation occurs. This may occur after one or two TC events. The counter is then disarmed automatically. The single or double TC requirement will depend on the state of other control bits. Note that even if the counter is automatically disarmed upon a TC, it always counts the count source edge which generates the trailing TC edge.

Bit CM6 specifies the location used to reload the counter contents when TC occurs. When CM6 = 0, the contents of the Load register are transferred into the counter at every TC. When CM6 = 1, the reload location may be either the Load or Hold register. The reload location in this case may be controlled externally using a GATE pin or may alternate on each TC. Bit CM7 controls the special gating functions that allow retriggering and the selection of Load and Hold locations for counter updating. The use and definition of CM7 will depend on the status of the Gating Control field and bits CM5 and CM6. See Figure 12.

Counter Mode bits CM13 through CM15 specify the hardware gating options. When "no gating" is selected (000) the counter will proceed unconditionally as long as it is armed. For any other gating mode, the count process is conditioned by the specified gating configuration. For codes of 110 or 111 in this field, counting proceeds after the specified active Gate edge occurs. Thereafter, the Gate input is ignored and counting continues until after one or two TC pulses occur or the counter is disarmed. Other codes in the gating field select either active-high or active-low level gating from a particular GATE pin, or from the TC signal of the adjacent counter. Level gating allows the counter to count only those clock edges that occur while the gate is active. Figure 12 summarizes the various counting configurations of the Am9513.

When edge gating is specified and the CM7 bit is cleared, counting will be enabled on the first active gate edge after the ARM instruction. Counting will continue until a DISARM instruction occurs or one or two TC pulses occur. When the counting stops on TC, an active GATE edge will allow counting to resume if the specified repetition has not occurred. While the counter is counting no GATE input edge or level will influence the count sequence. This mode provides a non-retriggerable, edge-triggered, digital one-shot function. When edge gating is specified and the CM7 bit is set, counting will begin on the active gate edge after the ARM instruction. If the specified repetition has not occurred, any active edge of the specified Gate input will reload the counter and at the same time the counter's contents will be transferred to the Hold register. In this mode the counter will stop after each TC and will resume on the next active GATE edge. Thus the counter performs as a retriggerable edge-triggered one shot. In the non-gated mode when the CM7 bit is

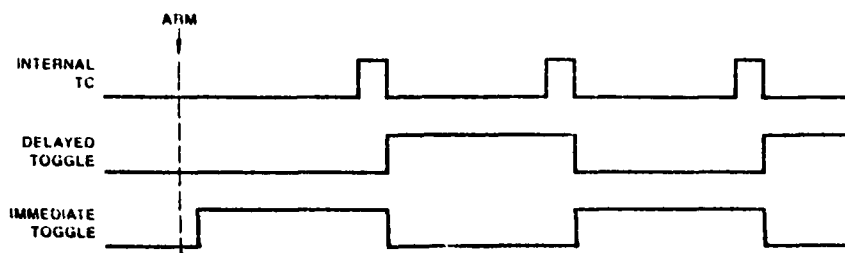


Figure 11. Output Waveforms.

MOS 178

Operating Mode	A	B	C	D	E	F	G	H	I	J	K	L
Special Gate (CM7)	0	0	0	0	0	0	0	0	0	0	0	0
Reload Source (CM6)	0	0	0	0	0	0	1	1	1	1	1	1
Repetition (CM5)	0	0	0	1	1	1	0	0	0	1	1	1
Gate Control (CM15-CM13)	000	LEVEL	EDGE	000	LEVEL	EDGE	000	LEVEL	EDGE	000	LEVEL	EDGE
Count to TC once, then disarm	X	X	X									
Count to TC twice, then disarm							X	X	X			
Count to TC repeatedly				X	X	X				X	X	X
Gate input does not gate counter input	X			X			X			X		
Count only during active gate level		X			X			X			X	
Start count on active gate edge and stop count on next TC			X			X						
Start count on active gate edge and stop count on second TC									X			X
No hardware retriggering	X	X	X	X	X	X	X	X	X	X	X	X
Reload counter from Load Register on TC	X	X	X	X	X	X						
Reload counter on each TC, alternating reload source between Load and Hold Registers							X	X	X	X	X	X
Transfer Load Register into counter on each TC that gate is LOW, transfer Hold Register into counter on each TC that gate is HIGH												
On active gate edge transfer counter into Hold register and then reload counter from Load Register												
On active gate edge transfer counter into Hold Register and then reload counter from Load or Hold Register												

Operating Mode	M	N	O	P	Q	R	S	T	U	V	W	X
Special Gate (CM7)	1	1	1	1	1	1	1	1	1	1	1	1
Reload Source (CM6)	0	0	0	0	0	0	1	1	1	1	1	1
Repetition (CM5)	0	0	0	1	1	1	0	0	0	1	1	1
Gate Control (CM15-CM13)	000	LEVEL	EDGE	000	LEVEL	EDGE	000	LEVEL	EDGE	000	LEVEL	EDGE
Count to TC once, then disarm		X	X									
Count to TC twice, then disarm							X	X	X			
Count to TC repeatedly					X	X				X	X	X
Gate input does not gate counter input							X			X		
Count only during active gate level		X			X			X			X	
Start count on active gate edge and stop count on next TC			X			X						
Start count on active gate edge and stop count on second TC									X			X
No hardware retriggering							X			X		
Reload counter from Load Register on TC		X	X		X	X						
Reload counter on each TC, alternating reload source between Load and Hold Registers								X	X		X	X
Transfer Load Register into counter on each TC that gate is LOW, transfer Hold Register into counter on each TC that gate is HIGH							X			X		
On active gate edge transfer counter into Hold Register and then reload counter from Load Register		X	X		X	X						
On active gate edge transfer counter into Hold Register and then reload counter from Load or Hold Register								X	X		X	X

Note: Operating modes M and P should not be used.

Figure 12. Am9513 Operating Modes.

set, the counter is enabled for frequency shift keying operation. In this mode, the counter's gate input will control whether the Load register or Hold register is reloaded into the counter at Terminal Count. The GATE input is synchronized with the count source to eliminate possible race conditions if it changes as the TC occurs.

When the Am9513 is set to operate with an 8-bit data bus width, pins DB8 through DB15 are not used for the data bus and are available for other functions. Pins DB13 through DB15 should be tied high. Pins DB8 through DB12 are used as auxiliary gating inputs, and are labeled GATE1A through GATE5A respectively. The auxiliary gate pin, GATENA, is logically ANDed with the gate input to Counter N, as shown in Figure 13. The output of the AND gate is then used as the gating signal for Counter N.

### Master Mode Register

The 16-bit Master Mode (MM) register is used to control all internal activities that are not controlled by the individual Counter Mode registers. This includes frequency control, time-of-day operation, comparator controls, data bus width and data pointer sequencing. Figure 14 shows the bit assignments for the Master Mode register.

A 16-bit scaling counter divides the output of the on-chip oscillator into four additional sub-frequencies. This provides a total of five internal frequencies that may be routed to any of the general counters and to the FOUT divider. The scaler is tapped every 4 bits and may be programmed by Master Mode bit MM15 to divide in binary or in BCD. Figure 15 shows the resulting combinations of frequencies that are available.

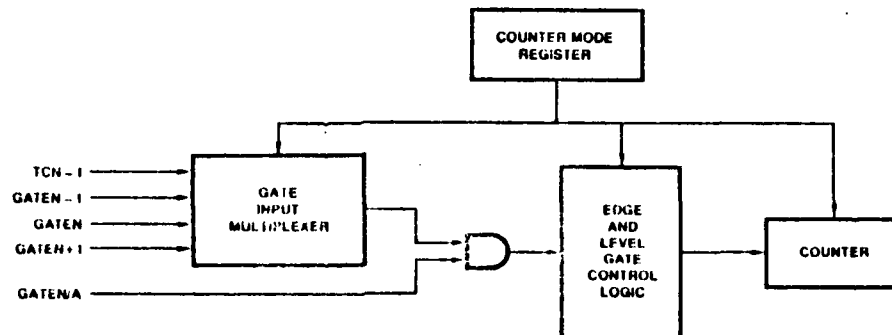


Figure 13. Gating Control.

MOS 179

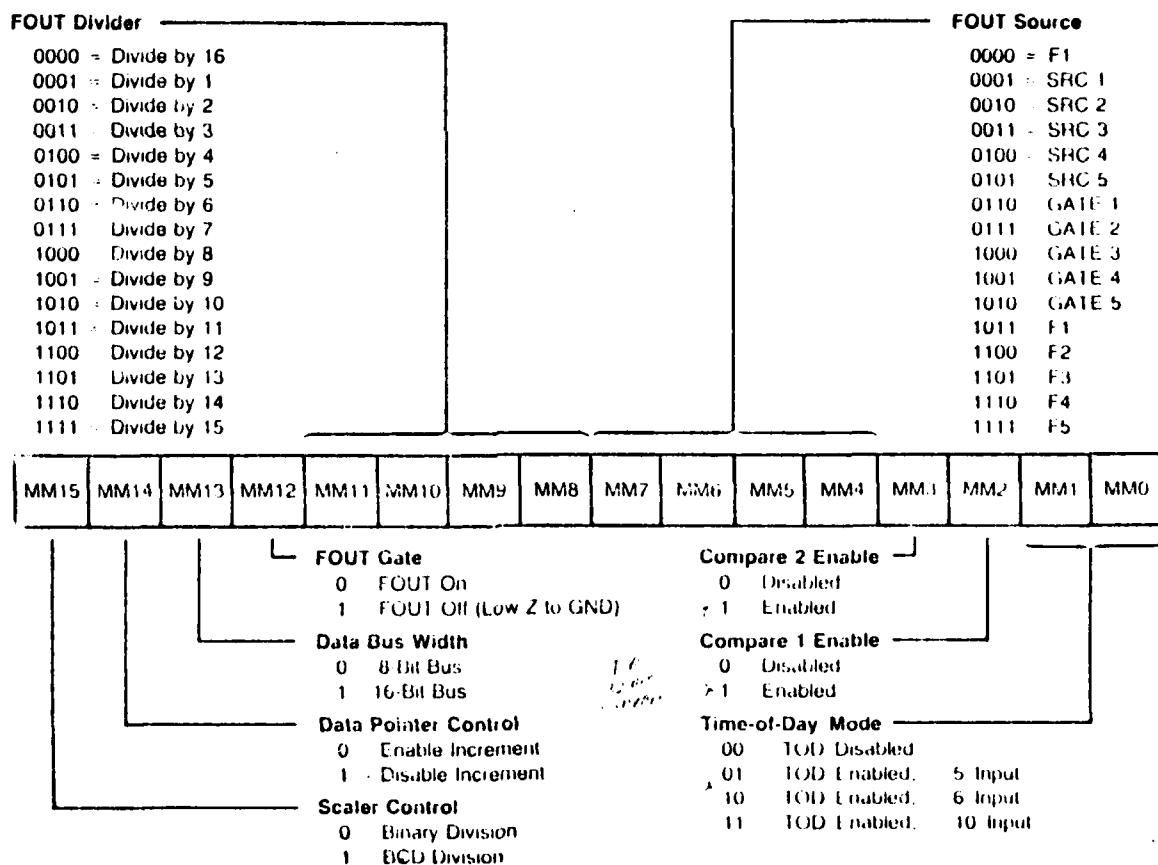


Figure 14. Master Mode Register Bit Assignments.

MOS 180

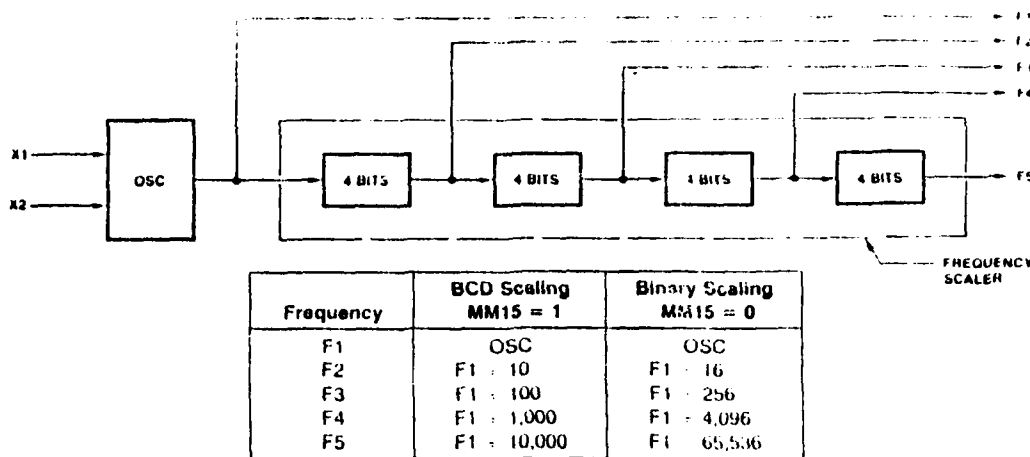


Figure 15. Internal Oscillator Frequency Scaler.

MOS 181

Bits MM0 and MM1 of the Master Mode register specify the time-of-day (TOD) options. When MM0 = 0 and MM1 = 0 the special logic used to implement TOD is disabled and counters 1 and 2 will operate in exactly the same way as counters 3, 4 and 5. When MM0 = 1 or MM1 = 1, additional counter decoding and control logic is enabled on counters 1 and 2 which causes their decades to turn over at the counts that generate appropriate 24-hour TOD accumulations.

Figure 16 shows the counter configurations for TOD operation. The least significant decade of Counter 1 is used to scale the input frequency in order to output tenth-of-second periods into the next decade. It can be setup to divide by five, divide by six, or divide by ten. Thus the input frequency for real-time clocking can be, respectively, 50Hz, 60Hz, or 100Hz. The input for Counter 2 should be the TC output of Counter 1 for TOD operation. Both counters should be setup for BCD counting and no gating. The Load registers should be used to initialize the clock to the proper time.

Added functions are available in the Counter Logic Groups for counters 1 and 2 (see Figure 2). Each contains a 16-bit Alarm register and a 16-bit Comparator. Bits MM2 and MM3 control the Comparators. When a Comparator is enabled its output is substituted for the normal counter output on the associated OUT1 or OUT2 pin. The polarity definition for the Comparator output will depend on the active-high or active-low definition as programmed in the appropriate Counter Mode register. Once the compare output is true, it will remain so until the count changes and the comparison therefore goes false. The two Comparators can be used individually in most operating modes. A special case occurs when the time-of-day option is invoked and both Comparators are enabled. The operation of Comparator 2 will then be conditioned by Comparator 1 so that a full 32-bit compare must be true in order to generate a true signal on OUT2.

Master Mode bits MM4 through MM7 specify the source input for the FOUT divider. Fifteen inputs are available for selection and they include the five Source pins, the five Gate pins and the five internal frequencies derived from the oscillator. The 5th combination of the four control bits (all zeros) is used to assure that an active frequency is available at the input to the FOUT divider following reset.

Bits MM8 through MM11 specify the dividing ratio for the FOUT Divider. The FOUT source (selected by bits MM4 through MM7) is divided by an integer value between 1 and

16, inclusive, and is then passed to the FOUT output buffer. After power-on or reset, the FOUT Divider is set to divide by sixteen.

Master Mode bit MM12 provides a software gating capability for the FOUT signal. When MM12 = 1, FOUT is off and in a low impedance state to ground. After power-up or reset, FOUT is gated on.

Bit MM13 controls the multiplexer at the data bus interface in order to configure the part for an 8-bit or 16-bit external bus. The internal bus is always 16-bits wide. When MM13 = 1, 16-bit data is transferred directly between the internal bus and all 16 of the external bus lines. In this configuration, the Byte Pointer bit in the Data Pointer register remains set at all times. When MM13 = 0, 16-bit internal data is transferred a byte at a time to and from the eight low-order external data bus lines. The Byte Pointer bit toggles with each byte transfer in this mode. When operating with an 8-bit data bus width, five of the eight high-order data bus pins (DB8 through DB12) are available for use as auxiliary gate inputs.

Bit MM14 controls the Data Pointer logic to enable or disable the automatic sequencing functions. When MM14 = 1, the contents of the Data Pointer can be changed only directly by entering a command. When MM14 = 0, several types of automatic sequencing of the Data Pointer are available. These are described in the Data Pointer register section of this document.

Bits MM12, MM13 and MM14 can be individually set and reset using commands issued to the Command register. In addition they can all be changed by writing directly to the Master Mode register.

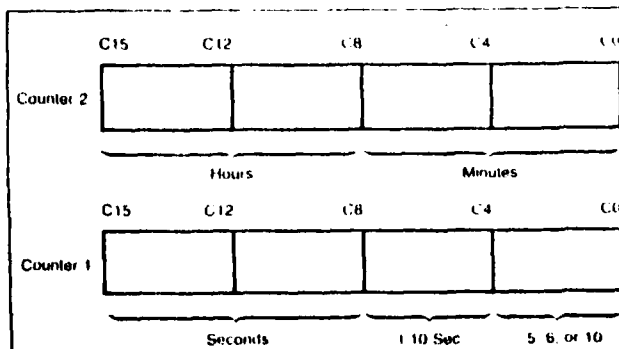


Figure 16. Time-of-Day Storage Configuration.

## APPLICATION INFORMATION

The X1 and X2 inputs can be driven with an RC network, an external TTL-level square wave, or a crystal. Figure 19 shows the suggested methods of connecting different frequency sources to the internal oscillator input.

The use of a crystal provides a highly accurate frequency source at moderate cost, and accordingly, will usually be the preferred method of operation. The Am9513 is designed to use a crystal in a parallel-resonant mode. The two ceramic capacitors connecting X1 and X2 to ground ensure proper loading on the crystal. The capacitor to X2 may be an adjustable type for fine-tuning the resonant frequency for critical applications.

An RC network provides a very low cost frequency source but may exhibit large frequency variations over recommended power supply and temperature ranges. Note that there is a resistor internal to the Am9513 in parallel with any external resistance.

### Initialization Procedures

The reset function in the Am9513 is accomplished in two ways: automatically during power-up and by software Master Reset command. Power-up reset circuitry is internally triggered by the rising VCC voltage when a predetermined threshold is reached. An internal flip-flop is set by the rising supply voltage and controls the reset operation. The reset flip-flop remains set until cleared by the first active Chip Select input. A reset may also be initiated by the host processor by entering the Master Reset

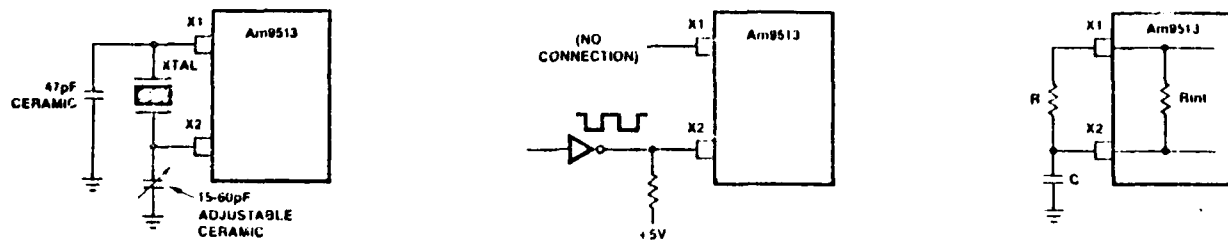
command. This software reset is active for the first read command after write, otherwise it performs the same function as a power-up reset.

Following either type of Reset, all five counters are disabled, 0B00 is loaded into each Counter Mode register, and 0000 is loaded in the Master Mode register. This results in each counter being configured to count down in binary on the positive-going edge of the internal F1 frequency source with no repetition or gating. The Master Mode register is cleared to configure the Am9513 for an 8-bit data bus width; binary division of the internal oscillator; FOUT gated on and set to divide F1 by 16; time-of-day mode and comparators 1 and 2 disabled; and the Data Pointer increment enabled.

Reset will clear the Load and Hold registers for each counter but will not change either the counter contents or the Data Pointer register.

The following initialization procedure should be followed on Counters 1 and 2 when Time-of-Day mode is selected

1. Set Time-of-Day enabled in the Master Mode register and load Counter Mode registers 1 and 2.
2. If Time-of-Day is to count up, load 0000 in Load registers 1 and 2 and execute command FF43 (Load) to load this value into the counters. This step conditions the count circuitry.
3. Load the desired start time into the Load registers and execute command FF43 again.
4. For counting up, load Load registers 1 and 2 with 0000.
5. Counters 1 and 2 may now be armed.



MOS 185

Figure 19. Driving the X1 and X2 Inputs.



# Low Cost, High Speed Data Acquisition Module

## DAS1128

### FEATURES

- Complete Data Acquisition System
- 12 Bit Digital Output
- 16 Single or 8 Differential Analog Inputs
- High Throughput Rate
- Selectable Analog Input Ranges
- Versatile Input/Output/Control Format
- Low 3 Watt Power Dissipation
- Small 3" x 4.6" x 0.375" Module



### GENERAL DESCRIPTION

The DAS1128 is a complete self-contained miniature high speed data acquisition system. The compact 3" x 4.6" x 0.375" module provides the designer with an easily implemented solution to the data acquisition problem. It contains an analog input signal multiplexer, a sample-and-hold amplifier, a 12 bit A/D converter, and all of the programming, timing and control circuitry needed to perform the complete data acquisition function.

The DAS1128 is a high performance device which can digitize an analog signal to an accuracy of  $\pm 1/2$  LSB out of 12 bits, relative to full scale. It has  $\pm 8$ ppm/ $^{\circ}$ C gain temperature coefficient, and the maximum throughput rate can be varied from 50,000 conversions/second for a 12 bit conversion from different analog channels, to 200,000 conversions/second for a successive 4 bit conversion made on a single channel.

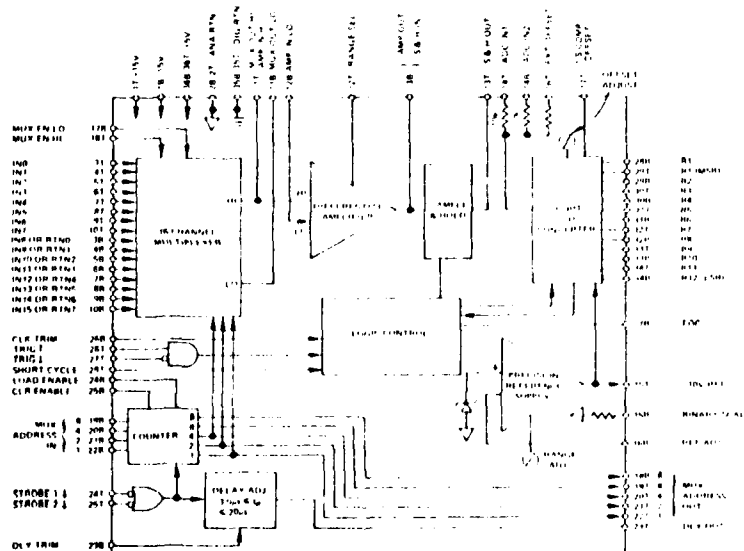


Figure 1. Functional Block Diagram

Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Analog Devices.

Route 1 Industrial Park; P.O. Box 280; Norwood, Mass. 02062  
Tel: 617/329-4700  
West Coast Mid-West Texas  
213/595-1783 312/894-3300 214/231-5094  
TWX: 710/394-6577

# SPECIFICATIONS

(typical @ +25°C and +15V unless otherwise noted)

## ANALOG INPUTS

Number of Inputs to Multiplexer	16 Single Ended, 8 True Differential, 16 Pseudo Differential
Input Voltage (Full Scale Range)	-10V to +10V, 0V to +10V, -5V to +5V, 0V to +5V, -10 24V to +10 24V, 0V to +10 24V, -5 12V to +5 12V, or 0V to +5 12V
Maximum Input Voltage	+15V
Input Current (per channel)	5nA max
Input Impedance	$>10^{10}$ ohms
Input Capacitance	10pF for "OFF" channel 100pF for "ON" channel
Input Fault Current (power off or MUX failure)	Internally limited to 20mA
Direct ADC Input Impedance	10k $\Omega$ for each input line

## ACCURACY<sup>1</sup>

Resolution	12 Bits
Error Relative to F.S.	$\pm 1/2$ LSB
Quantization Error	$\pm 1/2$ LSB
Differential Nonlinearity Error @ 33kHz throughput rate	$\pm 1/2$ LSB, 1LSB max
@ 50kHz throughput rate	$\pm 1$ LSB
Noise Error	$\pm 1/2$ LSB
-FS to +FS Error Between Successive Channel Transitions	$\pm 1$ LSB

## TEMP. COEFFICIENTS

Gain	8ppm/ $^{\circ}$ C, 20ppm/ $^{\circ}$ C max
Offset	5ppm/ $^{\circ}$ C, 15ppm/ $^{\circ}$ C max
Differential Nonlinearity	2.5ppm/ $^{\circ}$ C, 6ppm/ $^{\circ}$ C max

## SIGNAL DYNAMICS

Throughput Rate (12 Bits)	50kHz (max) (includes 5 $\mu$ s for MUX and SHA settling time plus 15 $\mu$ s for ADC)
MUX Crosstalk ("OFF" channels to "ON" channel)	$>80$ dB down @ 1kHz
Differential Amplifier CMRR	70dB to 1kHz
SHA Acquisition Time to 0.01%	4.5 $\mu$ s max
SHA Aperture Uncertainty	10ns
SHA Feedthrough	70dB down @ 1kHz

## DIGITAL INPUT SIGNALS

Compatibility	Standard DTL/TTL logic levels, 1 unit load/line
MUX Address Inputs (8, 4, 2, 1; Pins 19B through 22B)	Positive true natural binary coding selects channel for random addressing mode. Must be stable for 100ns after STROBE.
MUX ENABLE IH (Pin 18T)	High (Logic "1") input enables MUX "IH" output (for inputs 0 through 7)
MUX ENABLE LO (Pin 17B)	High (Logic "1") input enables MUX "LO" output (for inputs 8 through 15)
STROBE (Pin 24T or 25T)	Negative going transition (Logic "1" to Logic "0") updates MUX address register. STROBE 1 must be a Logic "1" to enable STROBE 2. STROBE 2 must be at Logic "1" to enable STROBE 1.
LOAD ENABLE (Pin 24B)	High (Logic "1") input allows next STROBE command to sequentially advance MUX address register.
	Low (Logic "0") input allows next STROBE command to update MUX address register according to external address inputs.
CLFAR ENABLE (Pin 25B)	Low (Logic "0") input allows next STROBE command to reset MUX address to channel "0" overriding LOAD ENABLE.
TRIGGER (Pin 26T)	Positive going transition (Logic "0" to Logic "1") initiates A/D conversion (even during conversion); TRIGGER (Pin 27T) must be at Logic "0" to allow TRIGGER function.
TRIGGER (Pin 27T)	Negative going transition (Logic "1" to Logic "0") initiates A/D conversion; Pin 26T (TRIGGER) must be at Logic "1" to allow TRIGGER function.

## DIGITAL OUTPUT SIGNALS

Compatibility	Standard DTL/TTL logic levels, 5 unit loads/line
Parallel Outputs	BE, BE through BE2
Coding	Natural binary, two's complement, offset binary, or one's complement
MUX Address Outputs (8, 8, 4, 2, 1; pins 18B, 19T through 22T)	Pin selectable
DELAY OUT (Pin 23T)	Positive true natural binary coding indicates channel selected

Negative going transition (Logic "1" to Logic "0") occurring normally 5 $\mu$ s (adjustable from 3 $\mu$ s to 20 $\mu$ s) after STROBE 1 command initiates A/D conversion automatically when connected to the TRIGGER. High (Logic "1") output during A/D conversion.

E $\ddot{O}$ C (Pin 27B)

## ADJUSTMENTS & TRIMS

Offset Adjust	
Internal Adjustment (Externally Accessible)	$\pm 10$ LSB's (min)
Remote External Adjustment (Pin 16T)	$\pm 10$ LSB's (min)
Range Adjust	
Internal Adjustment (Externally Accessible)	$\pm 10$ LSB's (min)
Remote External Adjustment (Pin 16B)	$\pm 10$ LSB's (min)
Clock Trim (Pin 26B)	
Factory Setting (Pin 26B "OPEN")	1.25 $\mu$ s/Bit
External Adjustment Range	1.25 $\mu$ s/Bit to 2.0 $\mu$ s/Bit
Delay Trim (Pin 23B)	
Factory Setting (Pin 23B "OPEN")	3 $\mu$ s
External Adjustment Range	3 $\mu$ s to 20 $\mu$ s

## CONTROLS

SHORT CYCLE (Pin 28T)	Connect to ground for full 12-bit resolution. Connect to B <sub>0</sub> output for resolution to B <sub>11</sub> 1-bit.
Channel Selection Mode (MUX Address Loading Mode)	Random, sequential continuous, and sequential triggered. Pin selectable.
A/D Conversion/Channel Select Sequences	Normal (input channel remains selected during its A/D conversion) and overlap (next channel selected during A/D conversion). Pin selectable.
Range Select (Pin 12T)	Differential Amplifier gain control connect to ANA RTN (Pin 21T) for X1 gain; connect to AMP OUT (Pin 13B) for X2 gain. This control is used in FSR selection procedure. Connect to REF ADJ (Pin 16B) to set reference to 10 24V. This control is used in FSR selection procedure, see Table II.
BINARY SCALE (Pin 15B)	Ground for 1's complement output code; connect to -15V dc for other available codes.
OUTPUT CODING (Pin 17T)	

## POWER REQUIREMENTS

+15V $\pm 3\%$	40mA, 50mA max
15V $\pm 3\%$	70mA, 100mA max
+5V $\pm 5\%$	250mA, 500mA max
Power Supply Sensitivity <sup>2</sup>	
Gain	$\pm 2.0$ mV/V
Offset	$\pm 4.0$ mV/V
Ref	$\pm 0.5$ mV/V

## ENVIRONMENT & PHYSICAL

Operating Temperature	0 to +70 $^{\circ}$ C
Storage Temperature	-25 $^{\circ}$ C to +85 $^{\circ}$ C
Relative Humidity	Up to 95% non condensing
Electrical Shielding	REF & FMI 6 sides (except connect or trace)
Packaging	Insulated steel cased module 3.00" x 4.60" x 0.375"

## PRICE

\$295.00 (1-9), price includes mating right angle connector

<sup>1</sup> Warmup time to rated accuracy is 5 minutes.

<sup>2</sup> Specification applies only when tracking +15V and -15V supplies are used, and for slowly occurring variations in power supply voltages.

Specifications subject to change without notice.

## THEORY OF OPERATION

A block diagram of the DAS1128 is shown in Figure 1. Analog input signals are applied to the various inputs of the 16 channel CMOS multiplexer. This multiplexer in conjunction with the differential amplifier that follows it, can be configured by the user to accept 16 single ended analog inputs, or 8 fully differential analog inputs. It can also be connected as a 16 channel "pseudo-differential" input device, which permits some of the benefits of differential operation while maintaining a 16 channel input capability.

The differential buffer amplifier is gain programmable by the user via jumpers at the module pins. This feature, along with the selectable reference voltages, permits the user to set up the DAS1128 to operate on any of 8 input voltage ranges. The differential amplifier drives a sample-and-hold amplifier, whose function it is to hold the selected analog input signal at a constant level while the A/D converter is making a conversion.

The A/D converter is a high speed 12 bit successive approximation device that has been designed using the Analog Devices' AD562, 12 bit integrated circuit D/A. The reference voltage for the conversion is supplied by an adjustable precision reference circuit that has a temperature coefficient of 5ppm/°C.

In addition to these basic functional blocks, the DAS1128 also contains all of the clock circuitry necessary to perform the complete data acquisition function. The internal clock can be externally adjusted to provide various throughput rates at different accuracies. Input channel addressing logic is provided, as is the capability to short cycle the A/D converter (i.e. perform conversions of less than 12 bits resolution). It is also possible for the user to adjust the time interval between input channel selection and the commencement of a conversion. The user can thus trade off speed vs. accuracy in the settling time of the multiplexer and sample-and-hold amplifier, as well as speed versus accuracy of the A/D converter.

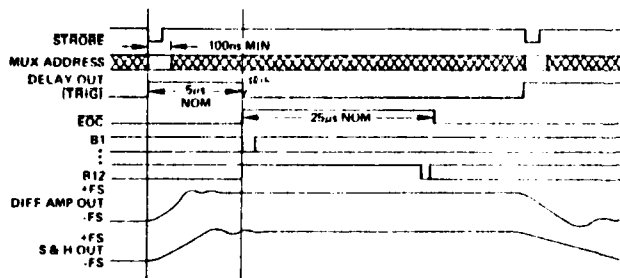


Figure 2. Simplified Timing Diagram, Showing Time-Interval Assignments and Constants

## INPUT CONNECTIONS

As shown in Figure 3, three input configurations can be used. 16 single-ended inputs (3a) can be connected to the multiplexer, all referenced to analog gnd. In the second configuration (3b), the inputs are connected individually as 8 true differential pairs. In this case the differential amplifier is connected "Differentially" with the output of the MUX. Finally, a "Quasi-Differential" connection (3c) can be realized under favorable ground path conditions. In this configuration the differential amplifier Lo terminal is used as the ground return

for all sensors. In each of these input schemes, it should be noted that the input multiplexer has been designed to protect itself and signal sources from both overvoltage failure and from fault currents due to power-off loading or MUX failure.

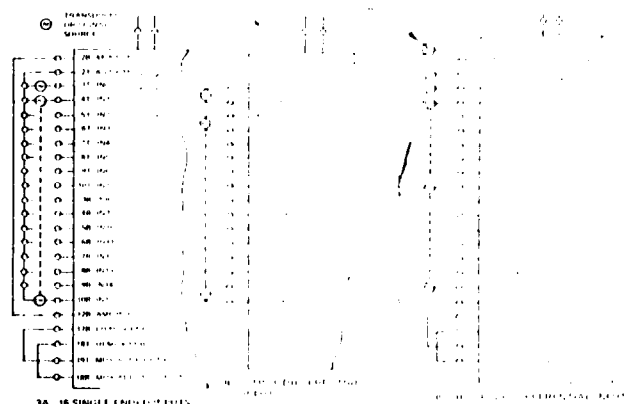


Figure 3. Signal Input Connections for Three Different Configurations

Full scale range of the DAS1128 may be set by appropriate jumper connections for 8 different ranges: 0 to +10V; 0 to +5V; 0 to +10.24V; 0 to +5.12V; -10 to +10V; -5 to +5V; -10.24 to +10.24V; -5.12 to +5.12V.

Note that 10.24 and 5.12 ranges are commonly used since conversion increments become 5mV/bit, 2.5mV/bit, and 1.25mV/bit.

## MUX AND S/H DYNAMICS - OVERLAP MODE

The overlap mode is defined as the ability of MUX to accept a new channel address thereby selecting the next channel to be sampled while the previously acquired sample is being held by the S/H for conversion. The dynamic characteristics of the S/H circuit are shown in Figure 4. Maximum throughput rates are obtainable when a single channel is held at a single address and the channel is sampled repeatedly. In a dynamic condition, data-throughput rates obtainable are shown in Figure 5.

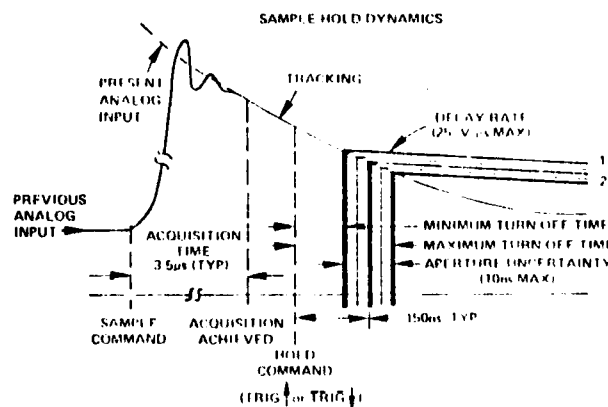


Figure 4. Sample-Hold Parameters Defined and Specified

## SHORT CYCLE

It is possible to short cycle the DAC1128, i.e., stop the conversion after less than 12 bits. This can be done by connecting an external jumper between short cycle terminal and one of the output terminals. With shorter cycles the attainable throughput rate increases, see Figure 5. In short cycle operation the EOC will decrease proportionately to the number of bits selected. Note the short cycle terminal *must* be grounded for full 12-bit operation.

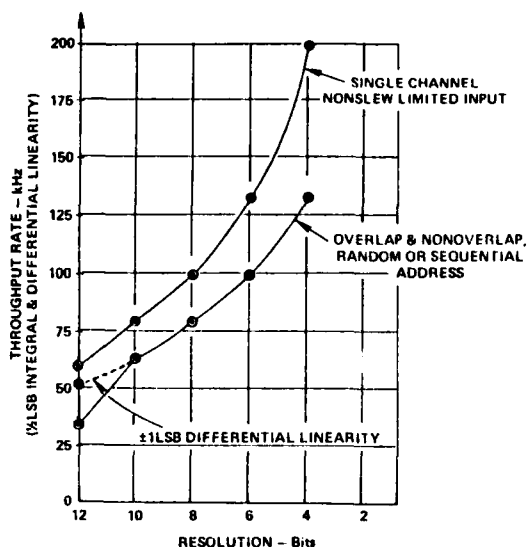
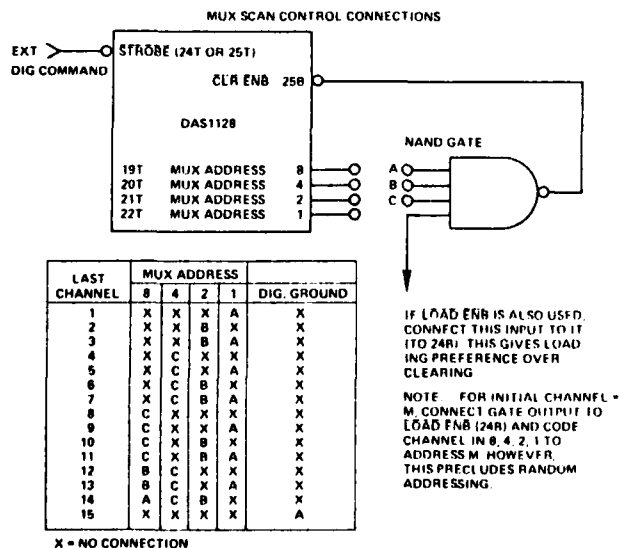


Figure 5. DAS1128 Throughput Rates

## MUX ADDRESSING

External terminals have been provided for the address counter. Thus the address counter can be configured to produce the following modes: Continuous sequential scanning (free running), sequential scanning with external step command, abbreviated scan continuously, random channel selection. See Figure 6 and set up procedure for details.



X = NO CONNECTION

Figure 6. To shorten scanning sequence of multiplexer channels, make the appropriate connections, (as shown in the chart) between an external NAND gate and MUX ADDRESS terminals 19T to 21T

## GROUPING CONSIDERATIONS

Attention should be given to the methods of connection for electrical returns and voltage reference points. Analog return (ANA RTN) and digital return (DIG RTN) are provided. The following rules should be applied when integrating the DAS1128 into the system.

1. If the  $\pm 15V$  power supply is floating (for optimum analog accuracy), connect its return to ANA RTN (Pin 2B or 2T). If the  $\pm 15V$  power supply is *not* floating, connect its return to DIG RTN (Pin 35T or 35B).
2. Connect the +5V supply return to DIG RTN (Pin 35T or 35B). If this supply also powers additional equipment, run separate, parallel returns to the equipment ground and to DIG RTN (Pin 35T or 35B).
3. To minimize signal grounding problems, single-ended input signals should only be returned to ANA RTN (Pin 2B or 2T). If this is not possible, then connect the input signals in either the "true differential" or "pseudo-differential" configurations (see Figure 3).
4. Connect computer ground to DIG RTN (Pin 35T or 35B). Use heavy wire or ground planes.
5. The computer chassis should be connected to the computer and power supply grounds at only one point.
6. Connect the third wire ground from main ac power input to the computer power supply return.

## GAIN AND OFFSET ADJUSTMENTS

The DAS1128 is calibrated with external gain and offset adjustment potentiometers connected as shown in Figure 7 and 8. The offset adjustment potentiometer has an adjustment range of at least  $\pm 10$ LSB's, and the gain range adjustment potentiometer has an adjustment range of at least  $\pm 10$ LSB's.

Offset calibration is not affected by changes in gain calibration, and should therefore be performed prior to gain calibration. Proper gain and offset calibration requires great care and the use of extremely sensitive and accurate reference instruments. The voltage standard used as a signal source must be very stable. It should be capable of being set to within  $\pm 1/10$ LSB of the desired value at any point within its range.

These adjustments are not made with zero and full scale input signals, and it may be helpful to understand why. An A/D converter will produce a given digital word output for a small range of input signals, the nominal width of the range being one LSB. If the input test signal is set to a value which should cause the converter to be on the verge of switching between two adjacent digital outputs, the unit can be calibrated so that it does switch at just that point. With a high speed convert command rate and a visual display, these adjustments can be performed in a very accurate and sensitive way. Analog Devices *Analog-Digital Conversion Notes* gives more detailed information on testing and calibrating A/D converters.

## OFFSET CALIBRATION

For unipolar +10V operation set the input voltage precisely to +0.0012V and adjust the offset potentiometer until the converter is just on the verge of switching from 000000000000 to 000000000001.

For  $\pm 5V$  bipolar operation set the input voltage precisely to -4.9988V; for  $\pm 10V$  units set it to -9.9976V. Adjust the offset

potentiometer, Figure 7, unit Offset Binary coded units are just on the verge of switching from 000000000000 to 000000000001 and Two's Complement coded units are just on the verge of switching 100000000000 to 100000000001.

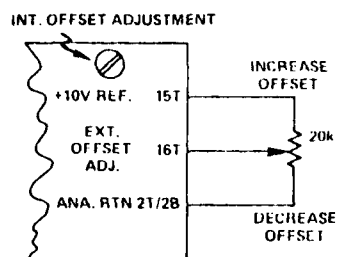


Figure 7. Ext. Offset Adjustment

### GAIN CALIBRATION

Set the input voltage precisely to +9.9963V for unipolar operation, +4.9963V for inputs of  $\pm 5V$  or +9.9926V for inputs of  $\pm 10V$ . Note that these values are  $1\frac{1}{2}$ LSB's less than nominal full scale. Adjust the 20k variable gain resistor, Figure 8, until Binary and Offset Binary coded units are just on the verge of switching from 11111111110 to 11111111111 and Two's Complement coded units are just on the verge of switching from 01111111110 to 01111111111.

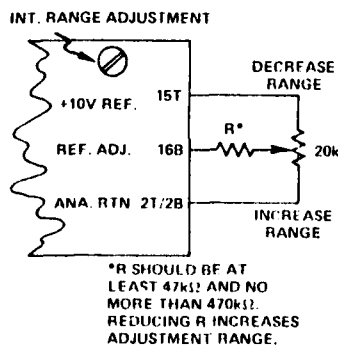


Figure 8. Ext. Ref. Adjustment

### CLOCK RATE ADJUSTMENT

The clock rate may be adjusted for best conversion time/accuracy trade off. The conversion time is varied by means of the external circuitry shown in Figure 9. An open CLK TRIM terminal (Pin 26B) results in 1.25 $\mu$ s/bit nominal conversion time. A grounded CLK TRIM terminal (for highest accuracy) results in 2.08 $\mu$ s/bit conversion.

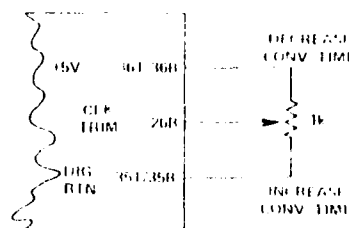


Figure 9. Clock Trim

### DELAY TIME ADJUSTMENT

The DLY OUT signal may be adjusted to vary the A/D converter triggering time by means of the external circuitry shown in Figure 10. An open DLY TRIM terminal (Pin 23B) results in a nominal delay time of 3.0 $\mu$ s. A grounded DLY TRIM terminal (for highest-accuracy) results in 20 $\mu$ s delay time nominal.

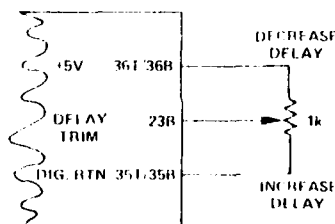


Figure 10. Delay Trim

TABLE I

INPUT CONFIGURATION	ANALOG INPUT CONNECTIONS	ANALOG INPUT RETURN	JUMPER CONNECTIONS
16 Single-Ended Inputs (Figure 3a)	3T thru 10T and 3B thru 10B	All input returns to 2B or 2T	11B to 11T 12B to 2B or 2T 17B to 19T 18T to 18B
8 Differential Inputs (Figure 3b)	3T thru 10T	3B thru 10B	11B to 12B 17B to 18T to "1"
16 Pseudo-Differential Inputs (Figure 3c)	3T thru 10T and 3B thru 10B	Common Input return to 12B	11B to 11T 17B to 19T 18T to 18B

## RECOMMENDED SET UP PROCEDURE

### 1. Select input configuration, see Table I.

### 2. Select MUX address mode.

The method of addressing the multiplexer can be selected by connecting the unit as follows:

**RANDOM.** Set Pin 24B ( $\overline{\text{LOAD ENB}}$ ) to Logic "0". The next falling edge of  $\overline{\text{STROBE}}$  will load the address presented to Pins 19B through 22B (8, 4, 2, 1). The code on these lines must be stable during the falling edge of  $\overline{\text{STROBE}}$  plus 100ns.

**SEQUENTIAL FREE RUNNING.** Set to Logic "1", Pin 24B ( $\overline{\text{LOAD ENB}}$ ) and 25B ( $\overline{\text{CLR ENB}}$ ). Connect Pin 27B ( $\overline{\text{EOC}}$ ) to Pin 24T ( $\overline{\text{STROBE}}$ ). Connect Pin 23T (DLY OUT) to Pin 27T (TRIG). Use Pin 26T (TRIG) as a run/stop control (i.e., A/D conversion will continue while TRIG is high and will stop while TRIG is low).

**SEQUENTIAL TRIGGERED.** Set to Logic "1", Pins 24B ( $\overline{\text{LOAD ENB}}$ ) and 25B ( $\overline{\text{CLR ENB}}$ ). Connect Pin 24T ( $\overline{\text{STROBE}}$ ) to external triggering source. The multiplexer address register will automatically advance by one channel whenever a  $\overline{\text{STROBE}}$  command is received. The initial channel can be selected by setting Pin 24B ( $\overline{\text{LOAD ENB}}$ ) to Logic "0" during only one  $\overline{\text{STROBE}}$  command. The multiplexer address will then be determined by the logic levels on Pins 19B through 22B (the external MUX address lines). Channel "0" can be selected as the initial channel by setting Pin 25B ( $\overline{\text{CLR ENB}}$ ) to Logic "0" during only one  $\overline{\text{STROBE}}$  command. The final channel can be selected by following the procedure presented in Figure 6.

### 3. Select A-D conversion/channel select sequence (see Figure 5).

- (1) **NORMAL** (input channel remains selected during its A/D conversion). Connect Pin 23T (DLY OUT) to Pin 27T (TRIG).
- (2) **OVERLAP** (next channel is selected during A/D conversion). Connect Pin 27B ( $\overline{\text{EOC}}$ ) to TTL compatible inverter input. Connect inverter output to Pin 24T ( $\overline{\text{STROBE}}$ ). Connect Pin 23T (DLY OUT) to Pin 27T (TRIG). Adjust the delay to at least  $4\mu\text{s}$  greater than  $\overline{\text{EOC}}$ ,  $20\mu\text{s}$  max (see Figure 10). The signal on Pin 26T (TRIG) serves as RUN/STOP control.
- (3) **REPETITIVE SINGLE CHANNEL.** After selecting the input channel to be repetitively sampled (see MUX ADDRESS MODE, above), set Pin 27T (TRIG) to Logic "0". Connect Pin 26T (TRIG) to a triggering source. Conversion process is initiated by positive edge of TRIG command.

### 4. Select output configuration.

- a. Full scale bit resolution: connect Pin 28T ( $\overline{\text{SHH CTR}}$ ) to Pin 35B (DLY RIN).
- b.  $B_n$  ( $B_n < 12$ ) bit resolution: connect Pin 28T to the output pin for  $B_n + 1$ .

### 5. Select optimum throughput rate.

The system clock frequency and the  $\overline{\text{STROBE}}$  to TRIG delay (if used) can be trimmed to optimize the accuracy/throughput rate trade-off. See Figures 9 and 10.

### 6. Select input voltage full scale range. See Table II.

### 7. Select output digital coding. See Table III.

TABLE II

FOR FULL SCALE RANGE OF:	MAKE THE FOLLOWING CONNECTIONS
0 to +10V	12T to 21; 14T to 14B to ADC Source*
0 to +10.24V	same as 0 to +10V, plus 15B to 16B.
0 to +5V	12T to 13B; 14T and 14B to ADC Source*
0 to +5.12V	same as 0 to +5V, plus 15B to 16B
-10V to +10V	12T to 21; 14T to 15T; and 14B to ADC Source*.
-10.24V to +10.24V	same as -10V to +10V, plus 15B to 16B
-5V to +5V	12T to 13B; 14T to 15T and 14B to ADC Source*.
-5.12V to +5.12V	same as -5V to +5V, plus 15B to 16B.

\*ADC Source is usually Sample and Hold Output (13T), but may be any signal source including Diff. Amp. Output (13B) if Sample and Hold is not desired.

TABLE III

OUTPUT CODE	CONNECTIONS
Unipolar Binary	Connect 17T to -15V Use 29T (B1) for MSB
2's Complement	Connect 17T to -15V Use 28B (B1) for MSB
Offset Binary	Connect 17T to -15V Use 29T (B1) for MSB
1's Complement	Connect 17T to 2B Use 28B (B1) for MSB

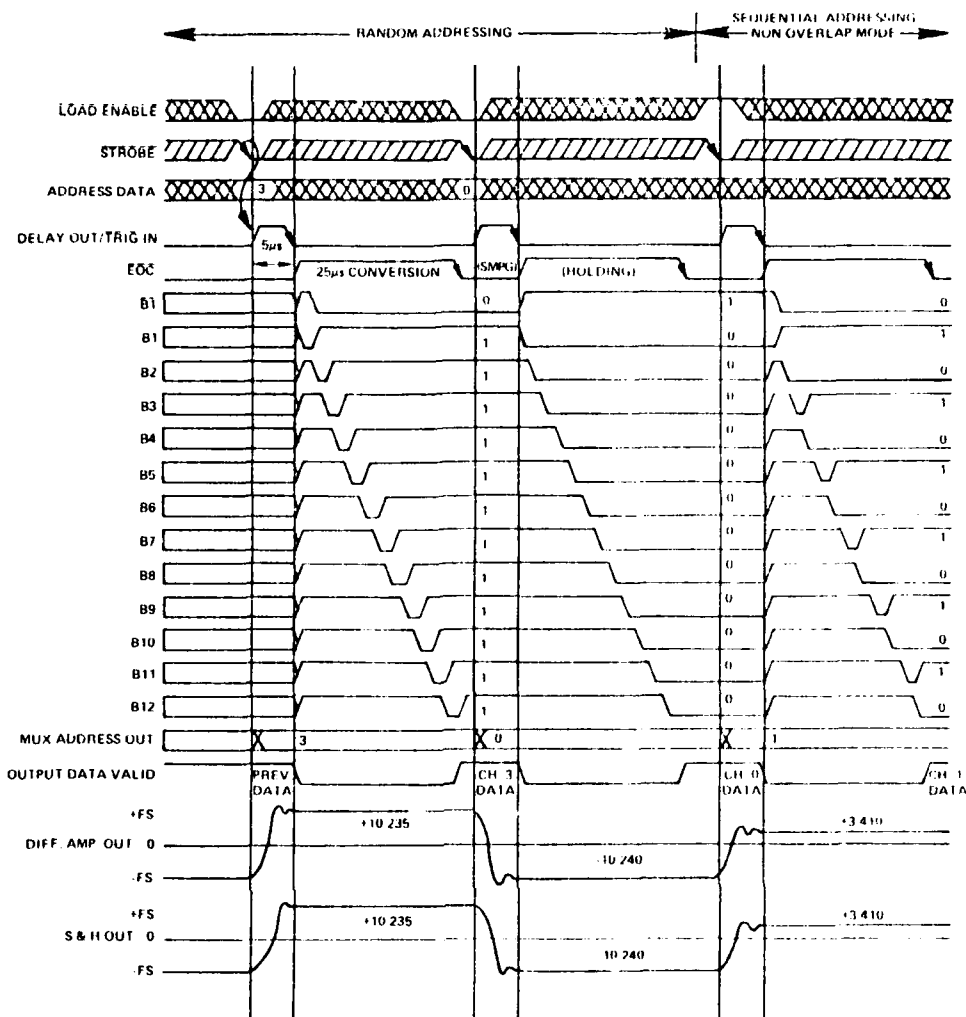


Figure 11. Timing for Non-Overlap Operation in Both Random and Sequential Addressing Modes. For Status Keys and Signal Condition Data, Refer to Box Below.

#### SIGNAL CONDITIONS AND STATUS KEYS FOR FIGURES 11 AND 12.

CH. 2 = -3.415V CODE 010 101 010 101  
CH. 3 = +10.235V CODE 111 111 111 111  
CH. 0 = -10.240V CODE 000 000 000 000  
CH. 1 = +3.410V CODE 101 010 101 010

ADC SET UP FOR  $\pm 10.24V$  INPUT,  
OFFSET BINARY. (FOR TWO'S  
COMPLEMENT, USE B1 FOR M.S.B.)

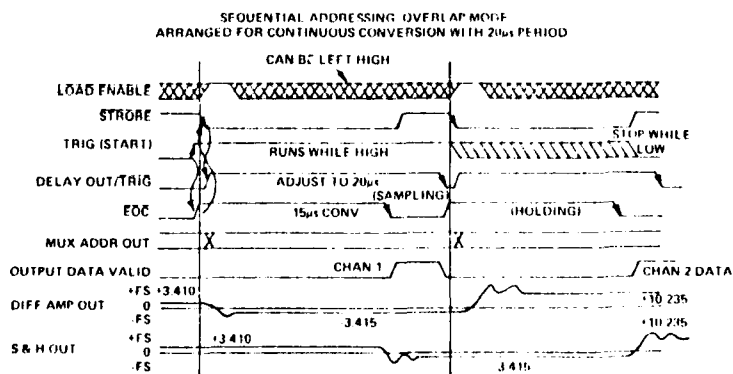


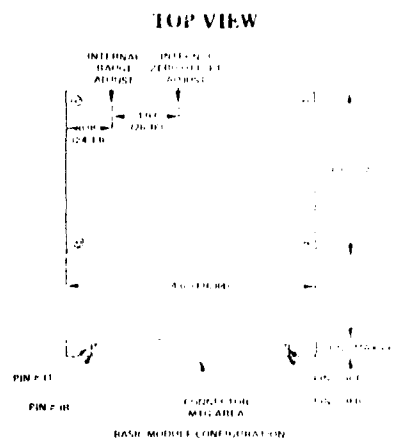
Figure 12. Timing Diagram for Overlap Operation in the Sequential Addressing Mode. For Status Keys and Signal Condition Data, See Box at Right.

KEY	INPUTS	OUTPUTS
	May change	Don't know
	May change 0 to 1	Changes 0 to 1
	May change 1 to 0	Changes 1 to 0
OR	Must be stable	Will be stable

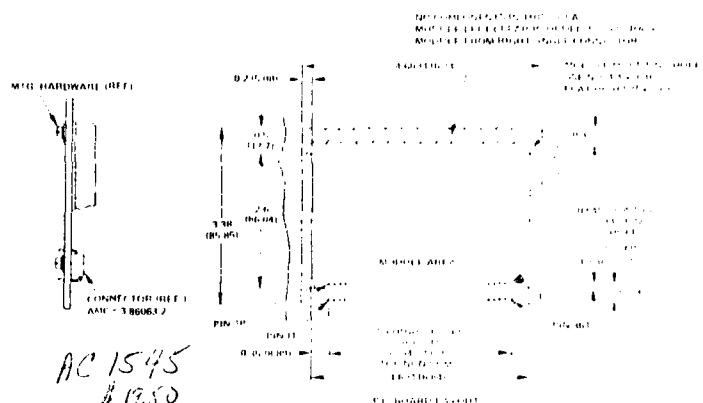
## Outline Drawings and Pin Designations

### DAS1128 Connector Pin Diagram

+15V	1T	1B	-15V	
ANA RTN	2T	2B	ANA RTN	
CH 0 IN	3T	3B	CH 8 IN	(CH 0 RTN)
CH 1 IN	4T	4B	CH 9 IN	(CH 1 RTN)
CH 2 IN	5T	5B	CH 10 IN	(CH 2 RTN)
CH 3 IN	6T	6B	CH 11 IN	(CH 3 RTN)
CH 4 IN	7T	7B	CH 12 IN	(CH 4 RTN)
CH 5 IN	8T	8B	CH 13 IN	(CH 5 RTN)
CH 6 IN	9T	9B	CH 14 IN	(CH 6 RTN)
CH 7 IN	10T	10B	CH 15 IN	(CH 7 RTN)
MUX HI OUT	11T	11B	MUX L0 OUT	
RANGE SEL	12T	12B	AMP IN LO	
S & H OUT	13T	13B	AMP OUT	
ADC IN 1	14T	14B	AUX IN 2	
+10V REF	15T	15B	BINARY SCALE	
EXT OFFSET	16T	16B	REF ADJ	
OUTPUT CODING	17T	17B	ENABLE LO	
ENABLE HI	18T	18B	8 OUT	
8 OUT 1	19T	19B	1 IN	
4 OUT 1	MUX ADDRESS	20T	20B	4 IN
2 OUT 1	LINES	21T	21B	2 IN
1 OUT 1		22T	22B	1 IN
DLY OUT		23T	23B	DLY TRIM
STROBE 1		24T	24B	LOAD TRN
STROBE 2		25T	25B	CLR FNR
TRIG		26T	26B	CLK TRIM
TRIG		27T	27B	EOC
SHT CYC		28T	28B	BT OUT
B1 OUT		29T	29B	DT OUT
B3 OUT		30T	30B	BA OUT
B5 OUT		31T	31B	BB OUT
B7 OUT		32T	32B	BB OUT
B9 OUT		33T	33B	B10 OUT
B11 OUT		34T	34B	B12 + B OUT
DIG RTN		35T	35B	DIG + IN
+5V	36T	36B	+5V	

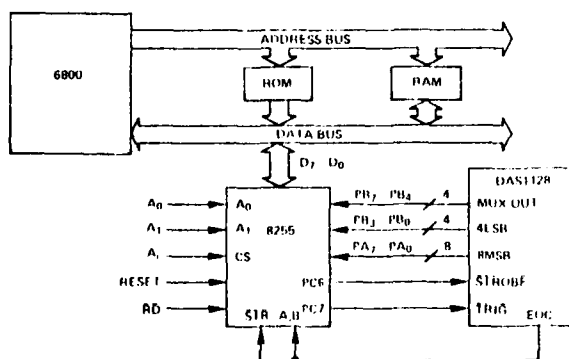


Dimensions shown in inches and (mm).



## Typical Applications

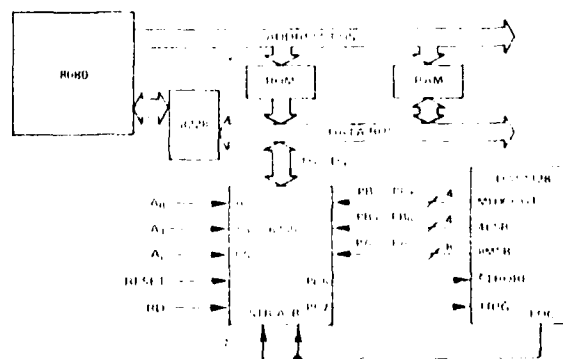
## DAS1128 WITH MOTOROLA 6800



**NOTE**

- 1 B255 USED IN MODE 1 (STORED I/O)
- 2 PC6 INDEXES MIX TO DESIRED CHANNEL
- 3 CS TO A, (WHERE A, IS AN ADDRESS BIT OTHER THAN  $A_0$  OR  $A_1$ )
- 4 PC7 INITIATES CONVERSION
- 5 FOC STORES IN DATA ANL MIX INFO
- 6 B255 SHOWN, HOWEVER 882) CAN ALSO BE USED

DAS1128 WITH INTEL 8080

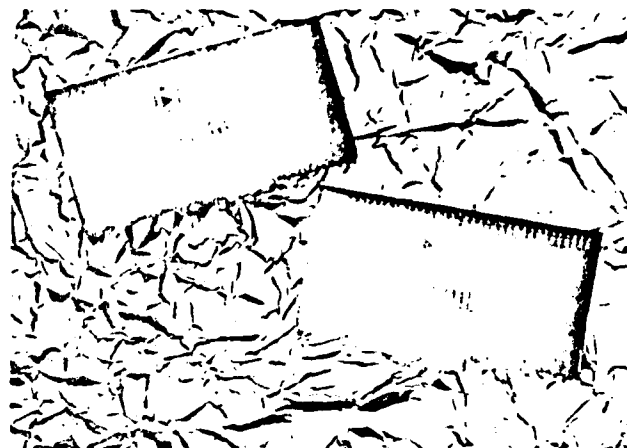


### NOTE

- 1 R255 USED TO MAKE TESTWORD 00
- 2 CS TO A, WHERE  $A_i$  IS AN ADDRESS NOT OTHER THAN  $A_0$  TO  $A_7$
- 3 PC6 INDICES MUX TO DESIRED CHANNEL
- 4 PC7 INITIATES CONVERSION
- 5 EUG STORES IN DATA AND MUX INFO

## FEATURES

- 12 Bit Resolution
- Input Register Included
- Choice of Codes
- Programmable Output Ranges
- Low Profile 2" x 4" x 0.4" Module



## GENERAL DESCRIPTION

The DAC1118 is a 12 bit, general purpose digital-to-analog converter which comes complete with an input storage register and a versatile output amplifier. This low profile 2" x 4" x 0.4" module has been designed to provide economical solutions to a wide range of digital-to-analog conversion problems. Performance specifications include 5 $\mu$ s settling time to 0.01%,  $\pm 20$ ppm/ $^{\circ}$ C gain temperature coefficient, and  $\pm 1/2$ LSB linearity error.

The design of the DAC1118 centers around the AD550 monolithic quad current switches and a hybrid resistor assembly consisting of matched precision resistors and a thick film network. The resistor assembly which contains all of the critical gain determining resistors and the quad switches with their inherent temperature tracking provide the DAC1118 with excellent performance over temperature.

The fully DTL/TTL compatible DAC1118 can be provided with a variety of input codes. In addition, any one of five voltage output ranges can be programmed by means of external jumpers connected to the module's terminal pins.

## DIGITAL INPUT CHARACTERISTICS

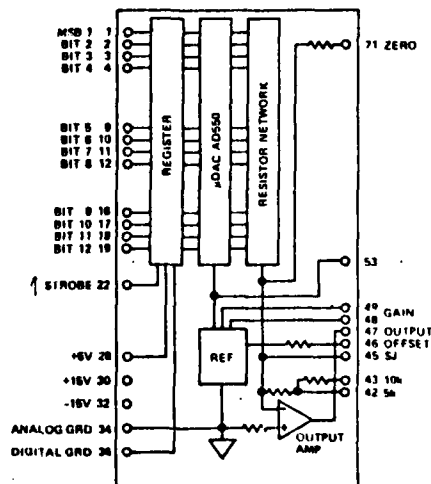
The TTL/DTL compatible storage register contained within the DAC1118 is configured during production to accept either Binary (including Offset Binary), Two's Complement, or BCD code. Digital data appearing at the converter's 12 input terminals will be strobed into the register whenever a positive-going transition is applied to the STROBE input (pin 22).

With the STROBE input held at either Logic "0" or Logic "1", the input data may be changed without affecting either the contents of the register or the output of the converter. The transfer characteristics of the DAC1118 are such that a full scale digital input (such as 1111...11 for Binary coded units) will result in a positive full scale voltage output.

## OUTPUT CHARACTERISTICS

The 12 binary-weighted current sources which form the basis of the digital to-analog conversion process are directly controlled by the digital data stored in the input register. The combined output of these sources is applied to the internal op amp summing junction to produce a voltage output signal. By connecting jumpers between the proper module pins, various values of op amp feedback resistance and thus, output voltage ranges can be selected.

In order to produce bipolar outputs, the current input to the internal op amp is offset by  $1/2$  Full Scale. This offset current is generated by the precision internal reference source and is applied to the op amp summing junction by means of jumpers connected to appropriate module terminals.



DAC1118 Block Diagram

# SPECIFICATIONS

(typical @ +25°C and rated supply voltage, unless otherwise specified)

RESOLUTION	12 Bits
DIGITAL INPUTS	
Logic Levels	0V < Logic "0" < 0.8V +2V < Logic "1" < +5V
Data Input Load	1 Standard TTL Load/bit
Strobe Input Load	3 Standard TTL Loads
Strobe Pulse Width	20ns (min)
Data Set-Up Time	20ns (min)
Data Hold Time	5ns (min)
INPUT CODES	
Unipolar	Binary, BCD
Bipolar	Offset Binary, 2's Complement
OUTPUT RANGES	0 to +5V @ 10mA 0 to +10V @ 5mA ±2.5V @ 10mA ±5V @ 10mA ±10V @ 5mA
OUTPUT IMPEDANCE	0.02Ω
SETTLING TIME	5μs to 0.01% <sup>1</sup>
Slewing Rate	20V/μs
LINEARITY ERROR	±1LSB (Max)
TEMPERATURE COEFFICIENT	
Gain	±20ppm/°C (±30ppm/°C max)
Zero	
Unipolar	±30μV/°C (±50μV/°C max)
Bipolar	±100μV/°C (±130μV/°C max)
Differential Linearity	±2ppm/°C (±10ppm/°C max)
TEMPERATURE RANGE	
Operating	0 to +70°C
Storage	-55°C to +100°C
POWER REQUIREMENTS	+15V ±5% @ 20mA (25mA max) -15V ±5% @ 30mA (45mA max) +5V ±10% @ 125mA (171mA max)
POWER SUPPLY SENSITIVITY <sup>2</sup>	
Gain	±20ppm/%V <sub>S</sub> of Reading <sup>3</sup>
Zero	±5ppm/%V <sub>S</sub> of Range <sup>3</sup>
DISDIMENTS	
User Provided)	
Gain (100kΩ, 20 Turn Pot)	±0.2% of Range
Zero (20kΩ, 20 Turn Pot)	±0.3% of Range

<sup>1</sup> For 10V step.

<sup>2</sup> For ±15V supplies only with ±15V and -15V supplies tracking.

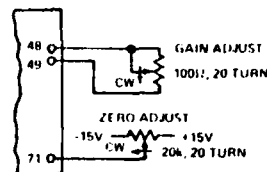
<sup>3</sup> Reading for bipolar input is defined as 1 Actual Reading - (- F.S.) L.

<sup>4</sup> Range for unipolar operation = + F.S.; Range for bipolar operation = 2 (- F.S.).

Specifications subject to change without notice.

## GAIN AND ZERO ADJUSTMENT

For units utilizing bipolar codes apply the digital input corresponding to the negative full scale analog output and adjust the zero pot until the value listed is obtained with ±1/10LSB. For Unipolar codes apply 00...0 and adjust for 0V out.



## Gain and Zero Pot Connections

For all units, once the appropriate zero adjustment has been made apply the digital input corresponding to the positive full scale analog output. Adjust the gain pot until the value listed is obtained within ±1/10LSB.

## OUTPUT CONNECTIONS

OUTPUT RANGE	PINS JUMPED TOGETHER			
	Bipolar Offset		Feedback Resistance	
	Binary	BCD	Binary	BCD
±2.5V	45, 46		47, 42	43, 45
±5.0V	45, 46		47, 42	
±10V	45, 46		47, 43	
0 to +5V	46, 34	46, 34	47, 42	43, 45
0 to +10V	46, 34	46, 34	47, 42	47, 43

## INPUT-OUTPUT RELATIONSHIPS

DIGITAL INPUT	NOMINAL VOLTAGE OUTPUT	
	0 to +5V Range	0 to +10V Range
Binary Code		
111111111111	+4.9988V	+9.9976V
000000000001	+0.0012V	+0.0024V
000000000000	0.0000V	0.0000V
BCD Code		
1001 1001 1001	+4.9950V	+9.9900V
0000 0000 0001	+0.0050V	+0.0100V
0000 0000 0000	0.0000V	0.0000V

## Unipolar Output

DIGITAL INPUT	NOMINAL VOLTAGE OUTPUT		
	±2.5V Range	±5V Range	±10V Range
Two's Complement Code			
011111111111	+2.4988V	+4.9976V	+9.9951V
000000000001	+0.0012V	+0.0024V	+0.0048V
000000000000	0.0000V	0.0000V	0.0000V
100000000000	-2.5000V	-5.0000V	-10.0000V
Offset Binary Code			
111111111111	+2.4988V	+4.9976V	+9.9951V
100000000001	+0.0012V	+0.0024V	+0.0048V
100000000000	0.0000V	0.0000V	0.0000V
000000000000	-2.5000V	-5.0000V	-10.0000V

## Bipolar Output

## ORDERING GUIDE:

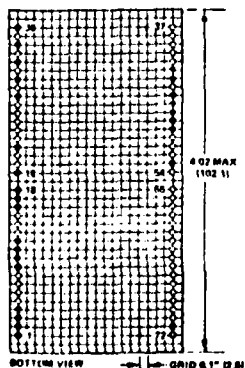
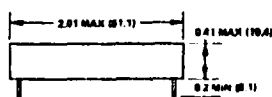
DAC1118 - XXX

1-2

- 023 (Binary, Offset Binary)
- 025 (BCD)
- 044 (2's Complement)

## OUTLINE DIMENSIONS

Dimensions shown in inches and (mm).



NOTE:  
Terminal pins installed only in shaded hole locations.  
Module weight: 3.5 ounces (99.3 grams)  
All pins are gold plated half-hard brass, (MIL-G-45214), 0.019" ±0.001" (0.48 ±0.03mm) dia.  
For plug in mounting card and connector 1500 (114.30mm) x 3.750" (95.25mm) 0.410" (10.41mm) order Board No. AC4494

MB64™ 64K STATIC RAM  
S-100 Bus

USER'S MANUAL

SSM MICROCOMPUTER PRODUCTS, INC.  
2190 Paragon Drive  
San Jose, California 95131

Telephone: (408) 946-7400  
TWX: 910-338-2077  
Telex: 171171  
DDD: (408) 946-3644 (110 Baud)

"MB64 is a trademark of SSM Microcomputer Products, Inc.

Written by Malcolm T. Wright  
Illustrated by Barbara R. Bastian

©1981 SSM MICROCOMPUTER PRODUCTS, INC.  
All Rights Reserved

Part No. MN0032  
January 1982

## TABLE OF CONTENTS

### 1.0 INTRODUCTION

### 2.0 SETTING UP YOUR MB64

- 2.1 Memory Addressing
- 2.2 Extended Addressing
- 2.3 Bank Select
  - 2.3.1 Bank Select Enable
  - 2.3.2 Bank Bit
  - 2.3.3 Power-Up State
- 2.4 EPROM Option
- 2.5 2K Memory Disable/Memory Organization
- 2.6 Board Enable Indicator
- 2.7 Battery Connector J1
- 2.8 Setup for CROMIX
- 2.9 Standard Setup
  - 2.9.1 64K memory, no options
  - 2.9.2 60K memory with top 4K EPROM
  - 2.9.3 Bottom 32K banked with top 32K master
  - 2.9.4 Two 32K banks, lower address
  - 2.9.5 48K banked slave memory
  - 2.9.6 64K memory with extended addressing

### 2.10 SWO Option

### 3.0 THEORY OF OPERATION

- 3.1 32K Memory Select PROM
- 3.2 Memory Address Select
- 3.3 2K Memory Decode
- 3.4 Magic Mapping
- 3.5 Extended Addressing
- 3.6 Bank Select Circuit
- 3.7 Battery Backup
- 3.8 Bus Interface

### 4.0 MEMORY TEST PROGRAM

### 5.0 TROUBLESHOOTING

- 5.1 Bank Select Test
- 5.2 Bank Preset
- 5.3 Memory Addressing

### 6.0 WARRANTY

APPENDIX:

Assembly Drawing  
Jumper Drawing  
Memory Map  
Parts List  
HM6116 Specification Sheet  
Schematic (INSERT)

## 1.0 INTRODUCTION

The SSM MB64 represents a significant advance in low-cost, high-density, static memory boards. It incorporates such features as bank select, extended addressing, diagnostic LEDs, and a provision for battery backup.

The MB64 is configurable as two 32K byte bank-switched memory blocks, or 64K bytes of memory with standard or extended addressing. The MB64 can be disabled in 2K increments by the use of a Magic Mapping™ circuit to provide memory space for other memory-mapped devices within the computer system. This board is equipped to sense the phantom disable line to prevent hardware conflicts with auxiliary system boot ROMs on another 696 compatible board.

### Features:

- Two 32K byte memory blocks
- Low power CMOS RAM chips (HM6116P)
- High speed access, approximately 150 nsec.
- Bank switching circuitry to support MP/M, CROMIX, OASIS, etc.
- Extended addressing to support IEEE 696 products
- LED indicators for RAM select and bank select
- Low power consumption of less than 300 ma typical
- Up to 8K of the top 32K can be replaced with EPROM (2716 type) on-board
- Card ejectors
- Goldplated PCB edge connector

The board has been designed to conform to the proposed IEEE 696 standard.

65,536 - 345,210  
65,536 - 527,277

---

Magic Mapping is a trademark of SSM Microcomputer Products, Inc.  
CROMIX is a trademark of Cromemco, 280 Bernard Avenue, Mountain View, CA.  
OASIS is a trademark of Phase One Systems, 7700 Edgewater Drive, Oakland, CA.  
MP/M is a trademark of Digital Research, P.O. Box 579, Pacific Grove, CA.

## 2.0 SETTING UP YOUR MB64

This section provides the information necessary to configure the MB64 to your particular application. Section 2.9 provides several example setups to assist you in performing this task. Be sure to also read Section 2.10, SWO OPTION, for proper read/write operation of the MB64 with your CPU.

**NOTE:** All of the following options (except BANK BIT) can be selected either by using supplied mini-jumpers or wire-wrapping.

### 2.1 MEMORY ADDRESSING

The MB64 is divided into two memory blocks of 32K bytes each. Block-A is physically located on the left half of the board; Block-B is physically located on the right half. Refer to the Memory Map in the APPENDIX for the address location of each memory chip.

Each 32K block can be addressed to either the lower or upper portion of the 64K memory space. Both 32K blocks can be addressed to the same address space if the MB64's Bank Select option is used and the bank bit for each block is different (refer to Section 2.3).

ADDRESS	BLOCK-B	BLOCK-A
Upper 32K	E17 to E18	E20 to E21
Lower 32K	E18 to E19	E21 to E22
Block Disabled	E18 open	E21 open

### 2.2 EXTENDED ADDRESSING

The MB64 is designed to support extended addressing (A16 thru A23) as defined in the proposed IEEE 696 standard. This will allow the MB64 to be placed on any 64K boundary within a maximum memory space of 16 megabytes.

The Extended Addressing option is enabled by connecting E26 to E27 for Block-A, and E29 to E30 for Block-B.

To select the extended address range which will enable the MB64, jumpers will be either installed or removed on the the header E1 thru E16 listed below.

Installing a jumper will provide a match when that particular address line is at a logic zero (low). Removing a jumper will provide a match when that particular address line is at a logic one (high).

Jumper installed = 0

Jumper removed = 1

ADDRESS (Hex)	A23	A22	A21	A20	A19	A18	A17	A16	ADDRESS RANGE
	E9 E1	E10 E2	E11 E3	E12 E4	E13 E5	E14 E6	E15 E7	E16 E8	
000000	0	0	0	0	0	0	0	0	1st 64K
010000	0	0	0	0	0	0	0	1	2nd 64K
020000	0	0	0	0	0	0	1	0	3rd 64K
030000	0	0	0	0	0	0	1	1	4th 64K
040000	0	0	0	0	0	1	0	0	5th 64K
FE0000	1	1	1	1	1	1	1	0	255th 64K
FF0000	1	1	1	1	1	1	1	1	256th 64K

When Extended Addressing is selected, Bank Select is disabled; therefore, Bank Select and Extended Addressing are **not possible together** for the same memory block.

## 2.3 BANK SELECT

To extend the amount of memory available for an 8-bit CPU, Bank Select can be used on the MB64. This memory management technique switches in and out a 64K bank of memory by writing to an I/O port. Multiple 32K blocks, on one or more MB64s, can be addressed to the same address space, but only **ONE** block will be active at a time. All bank control is done through port 40 Hex (or 41 Hex). When Bank Select is selected, extended addressing is disabled; therefore, bank select and extended addressing are not possible together on the same memory block.

### 2.3.1 Bank Select Enable

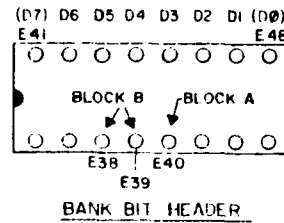
The two 32K blocks of memory on the MB64 can be individually bank selected. E26 thru E28 controls Block-A, while E29 thru E31 controls Block-B.

OPTION	BLOCK-A	BLOCK-B
Bank Select	E27 to E28	E30 to E31
Extended Addressing	E26 to E27	E29 to E30
Neither option	E27 open	E30 open

### 2.3.2 Bank Bit

One of 8 bits sent out to I/O port 40 Hex can be used to turn on (or off) a 64K bank of memory. If the data bit sent is a zero, the bank is disabled. If the data bit sent is a one, the bank is enabled.

All bank bit selection is through a 16 pin IC header. Pins E41 to E48 are data bits D7 thru D0, respectively. Pin E40 is for banking Block-A, and pin E38 & E39 are for banking Block-B.



Connect pins E38, E39 and E40 to the data bit desired by soldering a wire on the IC header. If only one bank bit is used for Block B, you can jumper E38 to E39. Be careful that pins E41 thru E48 **DO NOT SHORT TOGETHER**. Pins E41 thru E48 are the main S-100 bus' data output lines, and shorting them together may cause damage to other boards in your system.

### 2.3.3 Power-Up State

In a bank selected memory management system, one of the memory banks must be active on power-up to allow normal CPU operation. Each block on the MB64 can be set up to be enabled or disabled on reset (bus pin 75) of the computer.

POWER-UP STATE	BLOCK-A	BLOCK-B
Disabled	E35 to E36	E33 to E34
Enabled	E36 to E37	E32 to E33

If the Bank Select option is used on the MB64, the user **MUST** select one of the Power-Up options for each block. [NOTE: Reset must be generated on power-up of the computer during POC per the IEEE 696 standard.]

### 2.4 EPROM OPTION

The MB64 board is capable of supporting up to four 2716 EPROMs. Four IC sockets (U7, U14, U21, U28) have been provided with a jumper option to connect the Vpp (programming pulse) pin on the EPROM to +5 volts. The four IC sockets are addressed as the top 8K bytes of Block-B.

Address	RAM	ROM	Socket
1st 2K of 8K	E50 to E51	E49 to E50	U28
2nd 2K of 8K	E53 to E54	E52 to E53	U21
3rd 2K of 8K	E56 to E57	E55 to E56	U14
4th 2K of 8K	E59 to E60	E58 to E59	U7

[Remember that there are **NO** wait cycles generated by the MB64 for the EPROMs; therefore, the board cannot be run any faster than the EPROM's speed.]

## 2716 Manufacturers

NAME	PART	CURRENT		AVAILABLE SPEED
		STANDBY	READ	
Intel	2716	25ma (10ma Typ.)	100ma (57ma Typ.)	350ns to 650ns
NEC	UPD2716	25ma (10ma Typ.)	100 ma (57ma Typ.)	450ns
Motorola	MCM2716	25ma	100ma	250ns
AMI	S4716	25ma	100ma	450ns
Fujitsu	MBM2716	25ma	100ma	450ns
Hitachi	HN462716	35ma (21ma Typ.)	100ma (62ma Typ.)	450ns
National	NMC27C16	200ua	30ma (12ma Typ.)	450ns to 650ns

### 2.5 2K MEMORY DISABLE/MEMORY ORGANIZATION

The MB64 board is equipped with a special Magic Mapping circuit which allows the board to be disabled in 2K byte increments by simply removing the appropriate memory IC. This allows the user to free up memory space for memory-mapped devices such as disk interfaces, video boards, general I/O, ROM boards, etc.

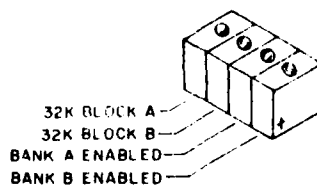
Magic Mapping relies on the **assumption** that the master CPU board has external pull-up resistors on the data input (DI) lines or the computer system has a **terminated bus**. The pull-up resistors on the DI bus will force the bus to "FF" Hex state, even if no S-100 board is present. Therefore, "FF" does not have to be transferred from the memory board to the data input bus. The code "FF" Hex is used internally by the MB64 to disable the ability to read the memory.

To determine which memory chip should be removed for a particular free address space, refer to the memory location map in the APPENDIX.

If the user wants to disable Magic Mapping for some reason, simply remove IC U44 (74LS30).

### 2.6 BOARD ENABLE INDICATOR

The MB64 has 4 memory state indicators at the top edge of the board. Two indicators show whether a 32K block of memory is selected. Two indicators show whether the bank for a block is enabled.



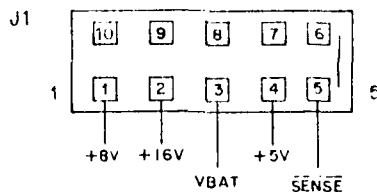
If the MB64 is set up for Bank Select (Section 2.3), both the block LED (label: ENA,ENB) and bank LED (label BANK A,BANK B) must be lit to read or write to the corresponding 32K bytes of memory.

## 2.7 BATTERY CONNECTOR J1

The MB64 is set up for a future battery backup piggyback board which will interface through connector J1. The connector J1 is used for:

- . Battery power input (VBAT)
- . Remote MB64 enable/disable (Sense)
- . Off-board logic power (+5V)
- . Battery charger power (+16V)
- . General purpose power (+8V)

When no battery backup is connected to the MB64, J1 pin 5 to pin 6 (sense) must be shorted together to enable the MB64. Also, J1 pin 3 to pin 4 must be connected to supply power to the memory chips when no battery power is available.



BATTERY CONNECTOR

## 2.8 SETUP FOR CROMIX

The MB64 can be set up to support the "User Memory" requirements of the CROMIX operating system by Cromemco. Under CROMIX, there can be up to 6 user memory boards of 64K bytes each. CROMIX requires bank-switched memory, with the upper 32K residing in the selected user space and also in the common memory bank (number 7).

The MB64 is split into two 32K banks called "A" and "B". Bank-A will be addressed to low memory range, while Bank-B will be addressed to the upper 32K range. Bank-B has two bank bit inputs (E38 & E39) so that it can be switched into two different banks per CROMIX. First, set up the MB64 for 64K with bank switching, and then select the bank bits for the user memory you are supporting. Typical setup is as follows:

### 64K Banked Slave Memory

CONNECT	COMMENT
E17 to E18	Enable upper 32K
E21 to E22	Enable lower 32K
J1-3 to J1-4	Enable memory power
J1-5 to J1-6	Enable MB64 (battery chip select)
E27 to E28	Enable Bank-A mode
E30 to E31	Enable Bank-B mode
E39 to E40	Set both bank bits the same
E35 to E36	Reset Bank-A to OFF
E33 to E34	Reset Bank-B to OFF
E24 to E25	Enable SW0 signal
E38 to E41	Enable upper 32K at Bank 7

### User Memory Setup

USER	CORRECT	CROMIX SIZE
1st	E40 to E47	One user system
2nd	E40 to E46	Two " "
3rd	E40 to E45	Three " "
4th	E40 to E44	Four " "
5th	E40 to E43	Five " "
6th	E40 to E42	Six " "

## 2.9 STANDARD SETUP

In this section we will try to show some standard configurations for the MB64.

### 2.9.1 64K memory, no options

CONNECT	COMMENT
E17 to E18	Enable upper 32K bytes
E21 to E22	Enable lower 32K bytes
J1-3 to J1-4	Enable memory power
J1-5 to J1-6	Enable MB64 (battery chip select)
E35 to E36	Reset Bank-A (turn off LED)
E33 to E34	Reset Bank-B (turn off LED)
E50 to E51	Set U28 as RAM
E53 to E54	Set U21 as RAM
E56 to E57	Set U14 as RAM
E59 to E60	Set U7 as RAM

AD-A155 465

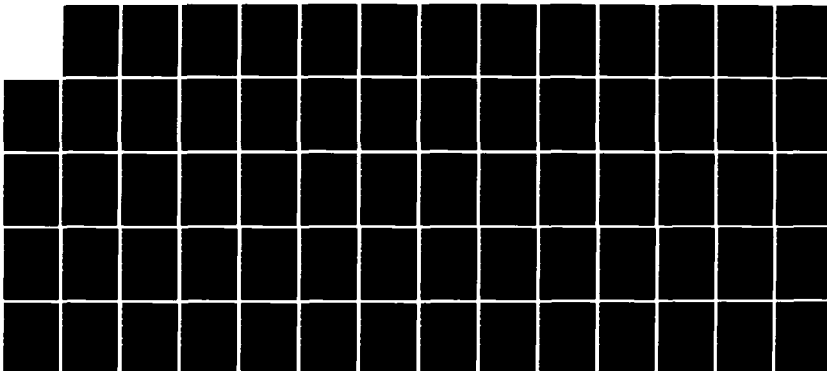
DEVELOPMENT OF A DEDICATED SPEECH WORK STATION(U) AIR  
FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF  
ENGINEERING W H LIEBER DEC 84 AFIT/GE/EE/84D-71

3/3

UNCLASSIFIED

F/G 9/2

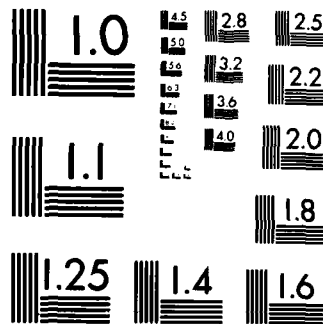
NL



END

11/11/84

11/11/84



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

### 2.9.2 60K memory with top 47 EPROM

CONNECT	COMMENT
E17 to E18	Enable upper 32K bytes
E21 to E22	Enable lower 32K bytes
J1-3 to J1-4	Enable memory power
J1-5 to J1-6	Enable MB64 (battery chip select)
E34 to E36	Reset Bank-A (turn off LED)
E33 to E34	Reset Bank-B (turn off LED)
E55 to E56	Set U14 as a ROM socket
E58 to E59	Set U7 as a ROM socket
E50 to E51	Set U28 as RAM
E53 to E54	Set U21 as RAM

Remove U7 and U14 RAM chips. Place first 2K of EPROM (2716) into socket U14. U14's socket is addressed at 0F000 Hex. Place second 2K of EPROM into socket U7. U7's socket is addressed at 0F800 Hex. Remember that the CPU's speed cannot be any greater than the access time of the on-board EPROMs unless wait states can be preset on the CPU or other 696 memory support boards.

### 2.9.3 Bottom 32K banked with top 32K master

CONNECT	COMMENT
E17 to E18	Enable upper 32K bytes
E21 to E22	Enable lower 32K bytes
J1-3 to J1-4	Enable memory power
J1-5 to J1-6	Enable MB64 (battery chip select)
E35 to E36	Reset Bank-A
E33 to E34	Reset Bank-B (turn off LED)
E27 to E28	Enable Bank-A mode
E40 to *	Select bank bit
E50 to E51	Set U28 as RAM
E53 to E54	Set U21 as RAM
E56 to E57	Set U14 as RAM
E59 to E60	Set U7 as RAM

\* Select bank bit per Section 2.3.2.

This setup makes the top 32K of memory a permanent (non-banked) master, while the lower 32K of memory is bank selected. The lower 32K of memory is switched off during reset or power-up, but can be turned on if E35 to E36 is changed to E36 to E37.

#### 2.9.4 Two 32K banks, lower address

CONNECT	COMMENT
E18 to E19	Enable lower 32K bytes (Block B)
E21 to E22	Enable lower 32K bytes (Block A)
J1-3 to J1-4	Enable memory power
J1-5 to J1-6	Enable MB64 (battery chip select)
E35 to E36	Reset Bank-A
E33 to E34	Reset Bank-B
E27 to E28	Enable Bank-A mode
E30 to E31	Enable Bank-B mode
E40 to *	Select bank bit for A
E39 to *	Select bank bit for B
E38 to E39	Strap up used input
E50 to E51	Set U28 as RAM
E53 to E54	Set U21 as RAM
E56 to E57	Set U14 as RAM
E59 to E60	Set U7 as RAM

\* Select bank bit per Section 2.3.2.

#### 2.9.5 48K banked slave memory

16K of memory (from Block-B) will be removed from the MB64 to make it a 48K only board.

CONNECT	COMMENT
E17 to E18	Enable upper 32K
E21 to E22	Enable lower 32K
J1-3 to J1-4	Enable memory power
J1-5 to J1-6	Enable MB64 (battery chip select)
E27 to E28	Enable Bank-A mode
E30 to E31	Enable Bank-B mode
E39 to E40	Set both bank bits the same
E38 to E39	Strap up used input
E38 to *	Select bank bit
E35 to E36	Reset Bank-A to OFF
E33 to E34	Reset Bank-B to OFF

\* Select bank bit per Section 2.3.2.

Remove U6, U7, U13, U14, U20, U21, U28 and U35 from their sockets. This will disable the top 16K of memory with the help of Magic Mapping.

### 2.9.6 64K memory with extended addressing

CONNECT	COMMENT
E17 to E18	Enable upper 32K
E21 to E22	Enable lower 32K
J1-3 to J1-4	Enable memory power
J1-5 to J1-6	Enable MB64 (battery chip select)
E35 to E36	Reset Bank-A LED
E33 to E34	Reset Bank-B LED
E26 to E27	Enable extended addressing (A)
E29 to E30	Enable extended addressing (B)
*E1 thru E16	Select extended addressing code
E50 to E51	Set U28 as RAM
E53 to E54	Set U21 as RAM
E56 to E57	Set U14 as RAM
E59 to E60	Set U7 as RAM

\* See Section 2.2 on Extended Addressing.

### 2.10 SWO OPTION

Some of the S-100 CPUs, and most of the IEEE 696 CPUs, are equipped with a write status signal called SWO. This signal can be used by the MB64 to control memory writing operations. If your CPU doesn't have this signal or the timing on this line is not correct, the SWO option can be disabled. Be sure to check your CPU for SWO (bus pin 97) and select the appropriate mode.

- Enable SWO for writing: Connect E24 to E25.
- Disable SWO for writing: Connect E23 to E24.

#### RECOMMENDED SETTINGS

Model	CPU	Manufacturer	Connect
CB1A	8080	SSM	E24 to E25
CB2	Z-80	SSM	E24 to E25
SCC	Z-80	Cromemco	E24 to E25
ZPU	Z-80	Cromemco	E24 to E25
ZPB	Z-80	Northstar	E23 to E24
-	Z-80	Dynabyte	E23 to E24
-	8080	WAMECO	E24 to E25
SBC-100	Z-80	SD Systems	E23 to E24
ZPU	Z-80	TDL	E24 to E25

### 3.0 THEORY OF OPERATION

#### 3.1 32K MEMORY SELECT PROM

The selection of 32K bytes of memory is controlled by a 256 x 4 PROM which replaces several discrete logic ICs by acting as a memory-mapped truth table of the logic functions desired. The address lines and chip select pins of the PROM are used as inputs to the logic function, while the output pins equal the truth table solution.

The input PROM signals are as follows:

SINP [bus 46] = PROM chip select 2 (E2)  
SOUT [bus 45] = PROM chip select 1 (E1)

BANK-A ENABLE	=	PROM, address A7
BANK-B ENABLE	=	" " A6
NOT ENABLE-A	=	" " A5
NOT ENABLE-B	=	" " A4
MAGIC MAPPING	=	" " A3
SMEMR [bus pin 47]	=	" " A2
PHANTOM [bus pin 67]	=	" " A1
PDBIN [bus pin 78]	=	" " A0

BANK-A or BANK-B ENABLE must be at a logic one to activate 32K of memory on the MB64. The NOT ENABLE lines are used to select the lower or upper 32K block of memory within a 64K boundary. The Magic Mapping line must be a logic one to enable a memory read. The PHANTOM line (per the proposed IEEE 696 standard) must be a logic one to read from or write into memory.

The output PROM signals are as follows:

32K SELECT-A	=	PROM data 3
32K SELECT-B	=	" " 2
OUTPUT ENABLE	=	" " 1
READ ENABLE	=	" " 0

The 32K SELECT lines from the PROM are used to control a 4-to-16 decoder which drives sixteen 2K RAM chips. The OUTPUT ENABLE line from the PROM goes to the  $\overline{OE}$  pin on each RAM. The READ ENABLE line from the PROM controls a tri-state buffer to transfer data from the selected memory chip to the data input (Di) bus in the computer.

To select a 32K block of memory, we need BANK ENABLE and ADDRESS (NOT ENABLE); therefore:

32K BLOCK-A = BANK-A ENABLE and not NOT ENABLE-A  
=  $A7 \cdot A5$

32K BLOCK-B = BANK-B ENABLE and not NOT ENABLE-B  
=  $A6 \cdot A4$

To select one 32K block that does not conflict with the other 32K block, both conditions **MUST NOT** be true. Also, PHANTOM disable must be true to select anything.

32K SELECT-A (D3) = 32K BLOCK-A and not 32K BLOCK-B and PHANTOM  
 =  $A7 \cdot \overline{A5} \cdot (A6 \cdot \overline{A4}) \cdot A1$

32K SELECT-B (D2) = 32K BLOCK-B and not 32K BLOCK-A and PHANTOM  
 =  $A6 \cdot A4 \cdot (A7 \cdot \overline{A5}) \cdot A1$

To provide an OUTPUT ENABLE signal to the RAMs, first the CPU must be performing a memory fetch operation (SMEMR) and the 32K must be selected.

OUTPUT ENABLE (D1) = SMEMR and (32K BLOCK-A or 32K BLOCK-B)  
 =  $A2 \cdot (A7 \cdot \overline{A5}) \oplus (A6 \cdot \overline{A4})$

To turn on the tri-state buffer (U43) to read memory, the read strobe (PDBIN) from the CPU must be true, as well as the Magic Mapping control line.

READ ENABLE (D0) = PDBIN and SMENR and MAGIC and PHANTOM  
 and (32K BLOCK-A or 32K BLOCK-B)

$$= A0 \cdot A2 \cdot A3 \cdot (A7 \cdot \overline{A5}) \oplus (A6 \cdot \overline{A4}) \cdot A1$$

With these four equations for the data lines of the PROM, the following truth table can now be generated for the PROM.

# TEST TABLE

ADDRESS (Hex)	A7	A6	A5	A4	A3	A2	A1	A0	D3	D2	D1	D0	DATA (Hex)	COMMENT
0	0	0	0	0	0	0	0	0	1	1	1	1	F	No condition met
1	0	0	0	0	0	0	0	1	1	1	1	1	F	
2	0	0	0	0	0	0	1	0	1	1	1	1	F	
3	0	0	0	0	0	0	1	1	1	1	1	1	F	
4	0	0	0	0	0	1	0	0	1	1	1	1	F	
5	0	0	0	0	0	1	0	1	1	1	1	1	F	
6	0	0	0	0	0	1	1	0	1	1	1	1	F	
7	0	0	0	0	0	1	1	1	1	1	1	1	F	
8	0	0	0	0	1	0	0	0	1	1	1	1	F	
9	0	0	0	0	1	0	0	1	1	1	1	1	F	
A	0	0	0	0	1	0	1	0	1	1	1	1	F	
B	0	0	0	0	1	0	1	1	1	1	1	1	F	
C	0	0	0	0	1	1	0	0	1	1	1	1	F	
D	0	0	0	0	1	1	0	1	1	1	1	1	F	
E	0	0	0	0	1	1	1	0	1	1	1	1	F	
F	0	0	0	0	1	1	1	1	1	1	1	1	F	
10	0	0	0	1	0	0	0	0	1	1	1	1	F	
11	0	0	0	1	0	0	0	1	1	1	1	1	F	
12	0	0	0	1	0	0	1	0	1	1	1	1	F	
13	0	0	0	1	0	0	1	1	1	1	1	1	F	
14	0	0	0	1	0	1	0	0	1	1	1	1	F	
15	0	0	0	1	0	1	0	1	1	1	1	1	F	
16	0	0	0	1	0	1	1	0	1	1	1	1	F	
17	0	0	0	1	0	1	1	1	1	1	1	1	F	
18	0	0	0	1	1	0	0	0	1	1	1	1	F	
19	0	0	0	1	1	0	0	1	1	1	1	1	F	
1A	0	0	0	1	1	0	1	0	1	1	1	1	F	
1B	0	0	0	1	1	0	1	1	1	1	1	1	F	
1C	0	0	0	1	1	1	0	0	1	1	1	1	F	
1D	0	0	0	1	1	1	0	1	1	1	1	1	F	
1E	0	0	0	1	1	1	1	0	1	1	1	1	F	
1F	0	0	0	1	1	1	1	1	1	1	1	1	F	
20	0	0	1	0	0	0	0	0	1	1	1	1	F	
21	0	0	1	0	0	0	0	1	1	1	1	1	F	
22	0	0	1	0	0	0	1	0	1	1	1	1	F	
23	0	0	1	0	0	0	1	1	1	1	1	1	F	
24	0	0	1	0	0	1	0	0	1	1	1	1	F	
25	0	0	1	0	0	1	0	1	1	1	1	1	F	
26	0	0	1	0	0	1	1	0	1	1	1	1	F	
27	0	0	1	0	0	1	1	1	1	1	1	1	F	
28	0	0	1	0	1	0	0	0	1	1	1	1	F	
29	0	0	1	0	1	0	0	1	1	1	1	1	F	
2A	0	0	1	0	1	0	1	0	1	1	1	1	F	
2B	0	0	1	0	1	0	1	1	1	1	1	1	F	
2C	0	0	1	0	1	1	0	0	1	1	1	1	F	
2D	0	0	1	0	1	1	0	1	1	1	1	1	F	
2E	0	0	1	0	1	1	1	0	1	1	1	1	F	
2F	0	0	1	0	1	1	1	1	1	1	1	1	F	

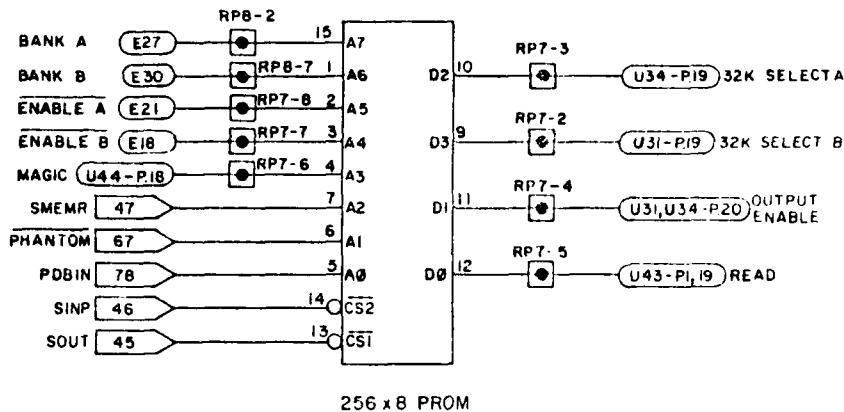
ADDRESS (Hex)	A7	A6	A5	A4	A3	A2	A1	A0	D3	D2	D1	D0	DATA (Hex)	COMMENT
30	0	0	1	1	0	0	0	0	1	1	1	1	F	
31	0	0	1	1	0	0	0	1	1	1	1	1	F	
32	0	0	1	1	0	0	1	0	1	1	1	1	F	
33	0	0	1	1	0	0	1	1	1	1	1	1	F	
34	0	0	1	1	0	1	0	0	1	1	1	1	F	
35	0	0	1	1	0	1	0	1	1	1	1	1	F	
36	0	0	1	1	0	1	1	0	1	1	1	1	F	
37	0	0	1	1	0	1	1	1	1	1	1	1	F	
38	0	0	1	1	1	0	0	0	1	1	1	1	F	
39	0	0	1	1	1	0	0	1	1	1	1	1	F	
3A	0	0	1	1	1	0	1	0	1	1	1	1	F	
3B	0	0	1	1	1	0	1	1	1	1	1	1	F	
3C	0	0	1	1	1	1	0	0	1	1	1	1	F	
3D	0	0	1	1	1	1	0	1	1	1	1	1	F	
3E	0	0	1	1	1	1	1	0	1	1	1	1	F	
3F	0	0	1	1	1	1	1	1	1	1	1	1	F	
40	0	1	0	0	0	0	0	0	1	1	1	1	F	
41	0	1	0	0	0	0	0	1	1	1	1	1	F	
42	0	1	0	0	0	0	1	0	1	0	1	1	B	Write Block-B
43	0	1	0	0	0	0	1	1	1	0	1	1	B	Write Block-B
44	0	1	0	0	0	1	0	0	1	1	0	1	D	
45	0	1	0	0	0	1	0	1	1	1	0	1	D	
46	0	1	0	0	0	1	1	0	1	0	0	1	9	Ready to read B
47	0	1	0	0	0	1	1	1	1	0	0	1	9	Ready to read B
48	0	1	0	0	1	0	0	0	1	1	1	1	F	
49	0	1	0	0	1	0	0	1	1	1	1	1	F	
4A	0	1	0	0	1	0	1	0	1	0	1	1	B	Write Block-B
4B	0	1	0	0	1	0	1	1	1	0	1	1	B	Write Block-B
4C	0	1	0	0	1	1	0	0	1	1	0	1	D	
4D	0	1	0	0	1	1	0	1	1	1	0	1	D	
4E	0	1	0	0	1	1	1	0	1	0	0	1	9	Ready to read B
4F	0	1	0	0	1	1	1	1	1	0	0	0	8	Read Block-B
50	0	1	0	1	0	0	0	0	1	1	1	1	F	
51	0	1	0	1	0	0	0	1	1	1	1	1	F	
52	0	1	0	1	0	0	1	0	1	1	1	1	F	
53	0	1	0	1	0	0	1	1	1	1	1	1	F	
54	0	1	0	1	0	1	0	0	1	1	1	1	F	
55	0	1	0	1	0	1	0	1	1	1	1	1	F	
56	0	1	0	1	0	1	1	0	1	1	1	1	F	
57	0	1	0	1	0	1	1	1	1	1	1	1	F	
58	0	1	0	1	1	0	0	0	1	1	1	1	F	
59	0	1	0	1	1	0	0	1	1	1	1	1	F	
5A	0	1	0	1	1	0	1	0	1	1	1	1	F	
5B	0	1	0	1	1	0	1	1	1	1	1	1	F	
5C	0	1	0	1	1	1	0	0	1	1	1	1	F	
5D	0	1	0	1	1	1	0	1	1	1	1	1	F	
5E	0	1	0	1	1	1	1	0	1	1	1	1	F	
5F	0	1	0	1	1	1	1	1	1	1	1	1	F	

ADDRESS (Hex)	A7	A6	A5	A4	A3	A2	A1	A0	D3	D2	D1	D0	DATA (Hex)	COMMENT
60	0	1	1	0	0	0	0	0	1	1	1	1	F	
61	0	1	1	0	0	0	0	1	1	1	1	1	F	
62	0	1	1	0	0	0	1	0	1	0	1	1	B	Write Block-B
63	0	1	1	0	0	0	1	1	1	0	1	1	B	Write Block-B
64	0	1	1	0	0	1	0	0	1	1	0	1	D	
65	0	1	1	0	0	1	0	1	1	1	0	1	D	
66	0	1	1	0	0	1	1	0	1	0	0	1	9	Ready to read B
67	0	1	1	0	0	1	1	1	1	0	0	1	9	Ready to read B
68	0	1	1	0	1	0	0	0	1	1	1	1	F	
69	0	1	1	0	1	0	0	1	1	1	1	1	F	
6A	0	1	1	0	1	0	1	0	1	0	1	1	B	Write Block-B
6B	0	1	1	0	1	0	1	1	1	0	1	1	B	Write Block-B
6C	0	1	1	0	1	1	0	0	1	1	0	1	D	
6D	0	1	1	0	1	1	0	1	1	1	0	1	D	
6E	0	1	1	0	1	1	1	0	1	0	0	1	9	Ready to read B
6F	0	1	1	0	1	1	1	1	1	0	0	0	8	Read Block-B
70	0	1	1	1	0	0	0	0	1	1	1	1	F	
71	0	1	1	1	0	0	0	1	1	1	1	1	F	
72	0	1	1	1	0	0	1	0	1	1	1	1	F	
73	0	1	1	1	0	0	1	1	1	1	1	1	F	
74	0	1	1	1	0	1	0	0	1	1	1	1	F	
75	0	1	1	1	0	1	0	1	1	1	1	1	F	
76	0	1	1	1	0	1	1	0	1	1	1	1	F	
77	0	1	1	1	0	1	1	1	1	1	1	1	F	
78	0	1	1	1	1	0	0	0	1	1	1	1	F	
79	0	1	1	1	1	0	0	1	1	1	1	1	F	
7A	0	1	1	1	1	0	1	0	1	1	1	1	F	
7B	0	1	1	1	1	0	1	1	1	1	1	1	F	
7C	0	1	1	1	1	1	0	0	1	1	1	1	F	
7D	0	1	1	1	1	1	0	1	1	1	1	1	F	
7E	0	1	1	1	1	1	1	0	1	1	1	1	F	
7F	0	1	1	1	1	1	1	1	1	1	1	1	F	
80	1	0	0	0	0	0	0	0	1	1	1	1	F	
81	1	0	0	0	0	0	0	1	1	1	1	1	F	
82	1	0	0	0	0	0	1	0	0	1	1	1	7	Write Block-A
83	1	0	0	0	0	0	1	1	0	1	1	1	7	Write Block-A
84	1	0	0	0	0	1	0	0	1	1	0	1	D	
85	1	0	0	0	0	1	0	1	1	1	0	1	D	
86	1	0	0	0	0	1	1	0	0	1	0	1	5	Ready to read A
87	1	0	0	0	0	1	1	1	0	1	0	1	5	Ready to read A
88	1	0	0	0	1	0	0	0	1	1	1	1	F	
89	1	0	0	0	1	0	0	1	1	1	1	1	F	
8A	1	0	0	0	1	0	1	0	0	1	1	1	7	Write Block-A
8B	1	0	0	0	1	0	1	1	0	1	1	1	7	Write Block-A
8C	1	0	0	0	1	1	0	0	1	1	0	1	D	
8D	1	0	0	0	1	1	0	1	1	1	0	1	D	
8E	1	0	0	0	1	1	1	0	0	1	0	1	5	Ready to read A
8F	1	0	0	0	1	1	1	1	0	1	0	0	4	Read Block-A

ADDRESS (Hex)	A7	A6	A5	A4	A3	A2	A1	A0	D3	D2	D1	D0	DATA (Hex)	COMMENT
90	1	0	0	1	0	0	0	0	1	1	1	1	F	
91	1	0	0	1	0	0	0	1	1	1	1	1	F	
92	1	0	0	1	0	0	1	0	0	1	1	1	7	Write Block-A
93	1	0	0	1	0	0	1	1	0	1	1	1	7	Write Block-A
94	1	0	0	1	0	1	0	0	1	1	0	1	D	
95	1	0	0	1	0	1	0	1	1	1	0	1	D	
96	1	0	0	1	0	1	1	0	0	1	0	1	5	Ready to read A
97	1	0	0	1	0	1	1	1	0	1	0	1	5	Ready to read A
98	1	0	0	1	1	0	0	0	1	1	1	1	F	
99	1	0	0	1	1	0	0	1	1	1	1	1	F	
9A	1	0	0	1	1	0	1	0	0	1	1	1	7	Write Block-A
9B	1	0	0	1	1	0	1	1	0	1	1	1	7	Write Block-A
9C	1	0	0	1	1	1	0	0	1	1	0	1	D	
9D	1	0	0	1	1	1	0	1	1	1	0	1	D	
9E	1	0	0	1	1	1	1	0	0	1	0	1	5	Ready to read A
9F	1	0	0	1	1	1	1	1	0	1	0	0	4	Read Block-A
A0	1	0	1	0	0	0	0	0	1	1	1	1	F	
A1	1	0	1	0	0	0	0	1	1	1	1	1	F	
A2	1	0	1	0	0	0	1	0	1	1	1	1	F	
A3	1	0	1	0	0	0	1	1	1	1	1	1	F	
A4	1	0	1	0	0	1	0	0	1	1	1	1	F	
A5	1	0	1	0	0	1	0	1	1	1	1	1	F	
A6	1	0	1	0	0	1	1	0	1	1	1	1	F	
A7	1	0	1	0	0	1	1	1	1	1	1	1	F	
A8	1	0	1	0	1	0	0	0	1	1	1	1	F	
A9	1	0	1	0	1	0	0	1	1	1	1	1	F	
AA	1	0	1	0	1	0	1	0	1	1	1	1	F	
AB	1	0	1	0	1	0	1	1	1	1	1	1	F	
AC	1	0	1	0	1	1	0	0	1	1	1	1	F	
AD	1	0	1	0	1	1	0	1	1	1	1	1	F	
AE	1	0	1	0	1	1	1	0	1	1	1	1	F	
AF	1	0	1	0	1	1	1	1	1	1	1	1	F	
B0	1	0	1	1	0	0	0	0	1	1	1	1	F	
B1	1	0	1	1	0	0	0	1	1	1	1	1	F	
B2	1	0	1	1	0	0	1	0	1	1	1	1	F	
B3	1	0	1	1	0	0	1	1	1	1	1	1	F	
B4	1	0	1	1	0	1	0	0	1	1	1	1	F	
B5	1	0	1	1	0	1	0	1	1	1	1	1	F	
B6	1	0	1	1	0	1	1	0	1	1	1	1	F	
B7	1	0	1	1	0	1	1	1	1	1	1	1	F	
B8	1	0	1	1	1	0	0	0	1	1	1	1	F	
B9	1	0	1	1	1	0	0	1	1	1	1	1	F	
BA	1	0	1	1	1	0	1	0	1	1	1	1	F	
BB	1	0	1	1	1	0	1	1	1	1	1	1	F	
BC	1	0	1	1	1	1	0	0	1	1	1	1	F	
BD	1	0	1	1	1	1	0	1	1	1	1	1	F	
BE	1	0	1	1	1	1	1	0	1	1	1	1	F	
BF	1	0	1	1	1	1	1	1	1	1	1	1	F	

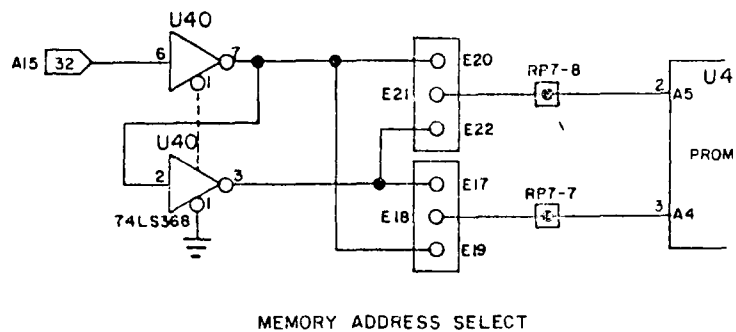
ADDRESS (Hex)	A7	A6	A5	A4	A3	A2	A1	A0	D3	D2	D1	D0	DATA (Hex)	COMMENT
C0	1	1	0	0	0	0	0	0	1	1	1	1	F	
C1	1	1	0	0	0	0	0	1	1	1	1	1	F	
C2	1	1	0	0	0	0	1	0	1	1	1	1	F	
C3	1	1	0	0	0	0	1	1	1	1	1	1	F	
C4	1	1	0	0	0	1	0	0	1	1	1	1	F	
C5	1	1	0	0	0	1	0	1	1	1	1	1	F	
C6	1	1	0	0	0	1	1	0	1	1	1	1	F	
C7	1	1	0	0	0	1	1	1	1	1	1	1	F	
C8	1	1	0	0	1	0	0	0	1	1	1	1	F	
C9	1	1	0	0	1	0	0	1	1	1	1	1	F	
CA	1	1	0	0	1	0	1	0	1	1	1	1	F	
CB	1	1	0	0	1	0	0	1	1	1	1	1	F	
CC	1	1	0	0	1	1	0	0	1	1	1	1	F	
CD	1	1	0	0	1	1	0	1	1	1	1	1	F	
CE	1	1	0	0	1	1	1	0	1	1	1	1	F	
CF	1	1	0	0	1	1	1	1	1	1	1	1	F	
D0	1	1	0	1	0	0	0	0	1	1	1	1	F	
D1	1	1	0	1	0	0	0	1	1	1	1	1	F	
D2	1	1	0	1	0	0	1	0	0	1	1	1	7	Write Block-A
D3	1	1	0	1	0	0	1	1	0	1	1	1	7	Write Block-A
D4	1	1	0	1	0	1	0	0	1	1	0	1	D	
D5	1	1	0	1	0	1	0	1	1	1	0	1	D	
D6	1	1	0	1	0	1	1	0	0	1	0	1	5	Ready to read A
D7	1	1	0	1	0	1	1	1	0	1	0	1	5	Ready to read A
D8	1	1	0	1	1	0	0	0	1	1	1	1	F	
D9	1	1	0	1	1	0	0	1	1	1	1	1	F	
DA	1	1	0	1	1	0	1	0	0	1	1	1	7	Write Block-A
DB	1	1	0	1	1	0	1	1	0	1	1	1	7	Write Block-A
DC	1	1	0	1	1	1	0	0	1	1	0	1	D	
DD	1	1	0	1	1	1	0	1	1	1	0	1	D	
DE	1	1	0	1	1	1	1	0	0	1	0	1	5	Ready to read A
DF	1	1	0	1	1	1	1	1	0	1	0	0	4	Read Block-A
E0	1	1	1	0	0	0	0	0	1	1	1	1	F	
E1	1	1	1	0	0	0	0	1	1	1	1	1	F	
E2	1	1	1	0	0	0	1	0	1	0	1	1	B	Write Block-B
E3	1	1	1	0	0	0	1	1	1	0	1	1	B	Write Block-B
E4	1	1	1	0	0	1	0	0	1	1	0	1	D	
E5	1	1	1	0	0	1	0	1	1	1	0	1	D	
E6	1	1	1	0	0	1	1	0	1	0	0	1	9	Ready to read B
E7	1	1	1	0	0	1	1	1	1	0	0	1	9	Ready to read B
E8	1	1	1	0	1	0	0	0	1	1	1	1	F	
E9	1	1	1	0	1	0	0	1	1	1	1	1	F	
EA	1	1	1	0	1	0	1	0	1	0	1	1	B	Write Block-B
EB	1	1	1	0	1	0	1	1	1	0	1	1	B	Write Block-B
EC	1	1	1	0	1	1	0	0	1	1	0	1	D	
ED	1	1	1	0	1	1	0	1	1	1	0	1	D	
EE	1	1	1	0	1	1	1	0	1	0	0	1	9	Ready to read B
EF	1	1	1	0	1	1	1	1	1	0	0	0	8	Read Block-B

ADDRESS (Hex)	A7	A6	A5	A4	A3	A2	A1	A0	D3	D2	D1	D0	DATA (Hex)	COMMENT
F0	1	1	1	1	0	0	0	0	1	1	1	1	F	
F1	1	1	1	1	0	0	0	1	1	1	1	1	F	
F2	1	1	1	1	0	0	1	0	1	1	1	1	F	
F3	1	1	1	1	0	0	1	1	1	1	1	1	F	
F4	1	1	1	1	0	1	0	0	1	1	1	1	F	
F5	1	1	1	1	0	1	0	1	1	1	1	1	F	
F6	1	1	1	1	0	1	1	0	1	1	1	1	F	
F7	1	1	1	1	0	1	1	1	1	1	1	1	F	
F8	1	1	1	1	1	0	0	0	1	1	1	1	F	
F9	1	1	1	1	1	0	0	1	1	1	1	1	F	
FA	1	1	1	1	1	0	1	0	1	1	1	1	F	
FB	1	1	1	1	1	0	1	1	1	1	1	1	F	
FC	1	1	1	1	1	1	0	0	1	1	1	1	F	
FD	1	1	1	1	1	1	0	1	1	1	1	1	F	
FE	1	1	1	1	1	1	1	0	1	1	1	1	F	
FF	1	1	1	1	1	1	1	1	1	1	1	1	F	



### 3.2 MEMORY ADDRESS SELECT

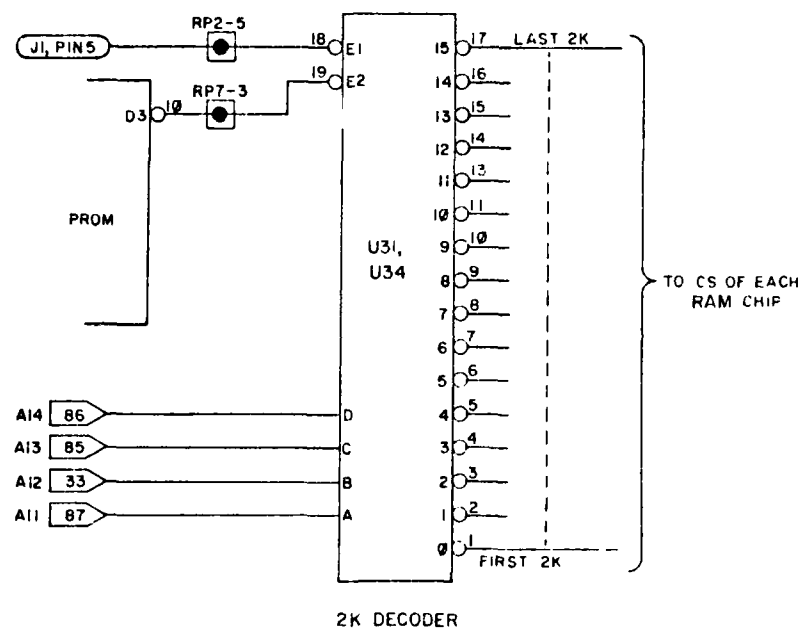
Memory address selection is provided by jumpers E17 thru E22. These jumpers allow the user to select the lower or upper 32K of memory within a 64K boundary by selecting the normal or inverted state of the A15 bus line. The input pins A4 & A5 must be a logic zero to enable 32K of memory.



### 3.3 2K MEMORY DECODE

After the input conditions are met on the PROM per Section 3.1, one of the two 32K select lines (D3 or D2) goes low, enabling a 4-to-16 decoder IC (74LS154). The 4-to-16 decoder receives address lines A11 thru A14 as its input and therefore will decode down to every 2K memory increment within one 32K boundary. Each of the 16 outputs of the 74LS154 goes to the chip select pin of a memory IC within a memory block.

The 4-to-16 decoder has two enabling pins. While one enable pin goes to 32K Select, the other enable pin goes to the battery back-up connector J1. J1, pin 5, is used by the battery back-up option to protect the data within RAM from being changed while on battery power. Without battery back-up, this pin is normally grounded to enable the 4-to-16 decoder.



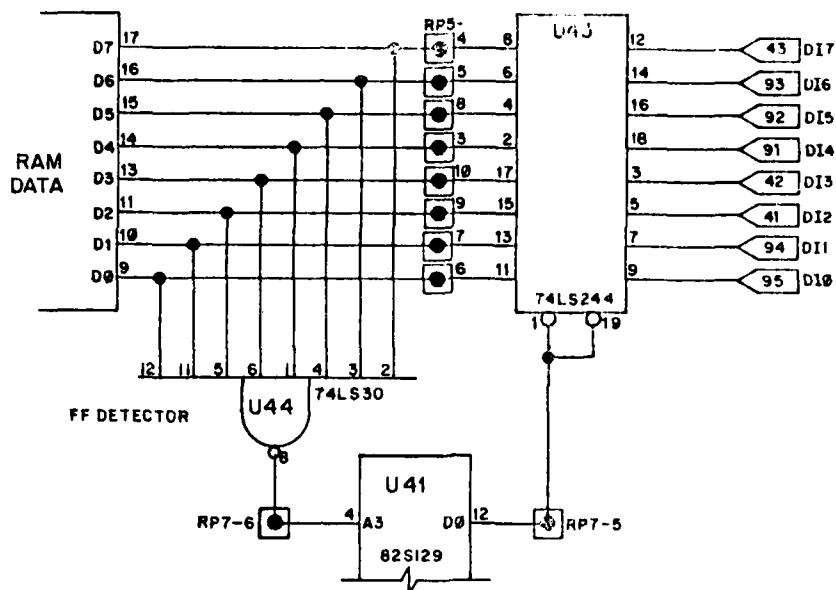
### 3.4 MAGIC MAPPING

Originally incorporated into the MESA in 1977, the Magic Mapping circuit allows a socket on the memory board and its supporting circuitry to be disabled by simply removing the IC chip. The Magic Mapping circuit disables the tri-state buffer from driving the data input (DI) bus if the memory IC is removed. Memory space (in 2K increments) can be made available for memory mapped video, I/O, disks or ROM boards within the 64K of memory.

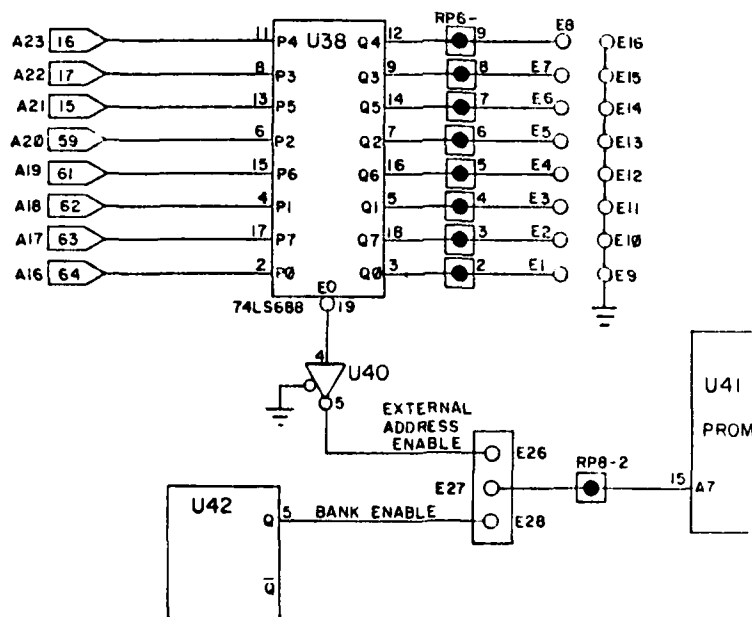
Magic Mapping relies on the **assumption** that the master CPU board has external pull-up resistors on the DI lines or the computer system has a **terminated bus**. The pull-up resistors on the DI bus will force the bus to "FF" Hex state, even if no S-100 board is present. Therefore, "FF" does not have to be transferred from the memory board to the DI bus.

U44 (74LS30) is an "FF" detector on the MB64. If an addressed memory chip puts out any Hex code except FF, then U44 outputs a one which will enable the read buffer (U43). If "FF" is detected, the output tri-state buffer is turned off, allowing any external S-100 board to drive the DI bus.

Magic Mapping allows for an entire 2K memory increment to be available for other memory boards or smaller areas by filling selected areas of RAM with "FF" Hex.



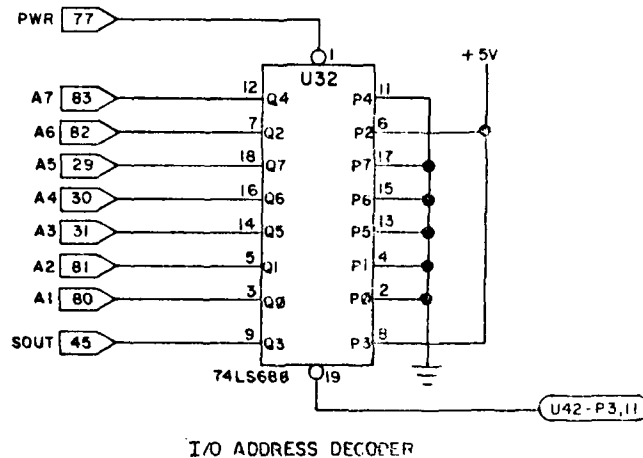
### 3.5 EXTENDED ADDRESSING



### 3.6 BANK SELECT CIRCUIT

The bank select function is driven by I/O port 40 or 41 Hex. This function is used in memory management schemes to provide multiple layers of memory, all addressed at the **SAME** address space. Each banked memory board is turned on or off by the memory manager software, for each task to be executed, as scheduled under time or priority interrupts.

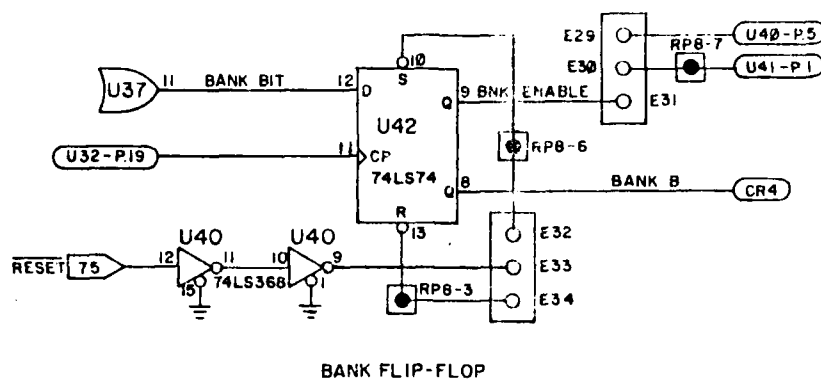
The I/O address decoder is one 74LS688 (U32) comparator. This IC puts out a negative pulse every time PWR, SOUT, and A1 thru A7 match the preset inputs on the other side of this comparator. Due to the lack of sufficient inputs to U32, A0 was not sensed, so the bank port address is 40 Hex and 41 Hex.



The bank select circuit is divided into two banks of 32K each. A bit within the byte sent to I/O port 40 Hex controls the activation of a bank. To save the user-selected bit for continuous bank control, a flip-flop (U41) is used for storage. The bit to be stored is selected by a 16-pin IC header. If the bit sent to the D-input of the flip-flop is a one, then the 32K block is enabled. If the bit sent is a zero, then 32K is disabled.

On power up in a bank-selected computer system, one memory bank is usually activated as the master. The flip-flop has been provided with jumper options to force the 32K block ON or OFF on power-up (POC) or reset. [NOTE: The proposed IEEE 696 standard requires Reset to be generated when the POC bus line is low.]

Jumpering <sup>22</sup>E32 to E34 (E35 to E37) connects the computer's reset signal to the set input or clear input of the bank flip-flop.



If jumper E32 to E33 (or E36 to E37) is connected, the memory block that flip-flop is controlling becomes a master bank on reset. If jumper E33 to E34 (or E35 to E36) is connected, the reset signal clears the "Q" output of the flip-flop and that memory block.

Two bank select indicators are provided on the MB64. Each flip-flop drives a LED to indicate if one of the 32K bank flip-flops are enabled.

### 3.7 BATTERY BACKUP

The MB64 is provided with a connector labelled J1 for an optional battery backup circuit. This battery option can maintain power to the memory chips for more than 5 hours, depending on the memory type and manufacturer used. For further information, please contact SSM for features and date of availability.

### 3.8 BUS INTERFACE

The MB64's memory chips are isolated from the S-100 bus by buffer ICs (74LS244, 74LS125, etc.) or general logic (74LS154, 82S129, etc.).

- a. Address lines A0 thru A7 are buffered by U33 (74LS244).
- b. Address lines A8 thru A10 are buffered by U39 (74LS244).
- c. Address lines A11 thru A14 are isolated by U31 & U34 (74LS154).
- d. Address line A15 is isolated by U40 (74LS368).
- e. Address lines A16 thru A23 are buffered by U38 (74LS688).
- f. Status lines SINP and SOUT are isolated by U41 (82S129).
- g. The status line for memory read (SMEMR) is isolated by U41 (82S129).
- h. The read strobe (PDBIN) is isolated by U41 (82S129).
- i. The write strobe ( $\overline{\text{PWR}}$ ) is isolated by U32 & U39 (74LS688 & 74LS125).
- j. The read/write disable line (Phantom) is isolated by U4 (82S129).
- k. The status line for writing (SWO) is isolated by U39 (74LS125).
- l. The memory data is read via U43 (74LS244).
- m. The memory data is written via U45 (74LS244).

The data buffer used during a read operation (U43) is controlled by the logic truth table within U41 (82S129) and issued on pin 12. U41, pin 12 will not provide a chip enable to U43 until the SMEMR, PDBIN, PHANTOM, A15, SINP and SOUT signals are in the correct state (see Section 3.1).

The data buffer used during a write operation (U45) is controlled by the buffer gate U39 (74LS125). To guarantee that the data is still present on the memory chip when the  $\overline{WE}$  signal goes high, the write line (U39, pin 11) drives the RAM and U45 directly. (U45 provides 10 nanoseconds or greater delay before the data becomes invalid at the end of a write cycle which meets the manufacturer's specification of 0 nanoseconds of data hold time.)

#### 4.0 MEMORY TEST PROGRAM

```

;      Simple Memory Test
;      Written by Andrew Schneider
;      Modified by Malcolm Wright
;      Coyright 1977 by SSM

;      Set "START" to the starting address of
;      memory to be tested. Set "MEND" to the last
;      address of memory to be checked.

;      The program will stop (HALT) when complete
;      or if an error was found. "GORB" (good or
;      bad) will be set to 00H for good memory or
;      to the byte pattern that would not read or
;      write correctly into memory. "LAST" is the
;      location where the last address tested will
;      be saved. If memory is good, then LAST=MEND.

8000 =      BEGIN EQU      8000      ;Start of program
E000 =      START EQU      0000H     ;Beginning of address
E3FF =      MEND EQU      7FFFH     ;Ending address

8000      ORG      BEGIN
8000 210000 LXI      H,START
8003 11FF7F LXI      D,MEND
8006 2B      DCX      H
8007 23      LOOP: INX      H
8008 3E7F     MVI      A,7FH
800A 07      CHECK: RLC
800B 77      MOV      M,A
800C BE      CMP      M
800D C22080   JNZ      ERROR
8010 B7      ORA      A
8011 FA0A80   JM       CHECK
8014 7B      MOV      A,E
8015 BD      CMP      L
8016 C20780   JNZ      LOOP
8019 7A      MOV      A,D
801A BC      CMP      H
801B C20780   JNZ      LOOP
801E 3E00     MVI      A,0
8020 322780   ERROR: STA      GORB      ;If using an IMSAI front panel
                                           ;replace with      CMA
                                           ;                      OUT OFFH
                                           ;to display byte on front panel.

8023 222880   SHLD     LAST
8026 76      HLT
8027 00      GORB: DB      0
8028 0000     LAST: DW      0
802A      END

```

## 5.0 TROUBLESHOOTING

Some checkout of the MB64 can be done by just watching the LEDs on the board.

### 5.1 BANK SELECT TEST

If you have another memory board which will run your system, you can temporarily disable the MB64 to test the banking circuitry.

- a. Remove jumpers from E17 to E22.
- b. Make a test header for E38 to E48.

Connect E40 to E48 (Bit0)  
Connect E39 to E47 (Bit1)  
Connect E38 to E46 (Bit2)

- c. Run the following routine (clear BANKS):

```
                ORG 100H
100 AF          XRA A      ; SET BANK BYTE=0
101 D3,40       OUT 40H    ; SEND BYTE
103 C3,00,00    JMP 0      ; GO BACK TO CP/M
```

All LEDs of the MB64 should not be lit. This checks both halves of U42 for a zero.

- d. Now run:

```
                ORG 100H
100 3E,01       MVI A,1    ; SET BANK BYTE=1
102 D3,40       OUT 40H    ; SEND BYTE
104 C3,00,00    JMP 0      ; GO BACK TO CP/M
```

Only the LED labelled BNKA should be lit. This checks one-half of U42.

- e. Now run:

```
                ORG 100H
100 3E,02       MVI A,2    ; SET BANK BYTE=2
102 D3,40       OUT 40H    ; SEND BYTE
104 C3,00,00    JMP 0      ; GO BACK TO CP/M
```

Only the LED labelled BNKB should be lit. This checks the other half of U42 and one input of U37.

- f. Now run:

```
                ORG 100H
100 3E,04       MVI A,4    ; SET BANK BYTE=4
102 D3,40       OUT 40H    ; SEND BYTE
104 C3,00,00    JMP 0      ; GO BACK TO CP/M
```

Only the LED labelled BNKB should be lit. This checks the other input of U37.

g. Last, run:

```
                ORG 100H ; SELECT CROSSED BANK
100 3E,08      MVI A,8   ; SET BANK BYTE=8
102 D3,40      OUT 40H   ; SEND BYTE
104 C3,00,00   JMP 0     ; GO BACK TO CP/M
```

No LEDs on the MB64 should be lit.

## 5.2 BANK PRESET

If you have another memory board which will run your system, you can temporarily disable the MB64 to test the banking circuitry.

- a. Remove jumpers from E17 to E22.
- b. Install jumpers E35 to E36 and E33 to E34.
- c. Push computer Reset. (DON'T LET THE SYSTEM BOOT.)

No LEDs on the MB64 should be lit. This tests the reset inputs of U42.

- d. Install jumpers at E36 to E37 and E32 to E33 now.
- e. Push computer Reset.

Both LEDs on the MB64 labelled BNKA & BNKB should be lit. This tests the set inputs of U42.

- f. Install jumpers at E36 to E37 and E33 to E34.
- g. Push computer Reset.

One LED on the MB64 labelled BNKA should be lit. This tests that the set/reset inputs are not shorted between the halves of U42.

## 5.3 MEMORY ADDRESSING

If the MB64 is set up in one of the many standard configurations indicated in Section 2.9, the LED's ENA or ENB should be flashing dimly as the computer accesses the board. If there are no jumpers on E17 thru E22, the MB64 cannot be read.

By enabling half of the MB64 (32K), it should be possible to run user-defined memory tests. Jumper E18 to E19 to test memory block B as the first 32K and leave E21 open. Jumper E21 to E22 to test memory block A as the first 32K and leave E18 open.

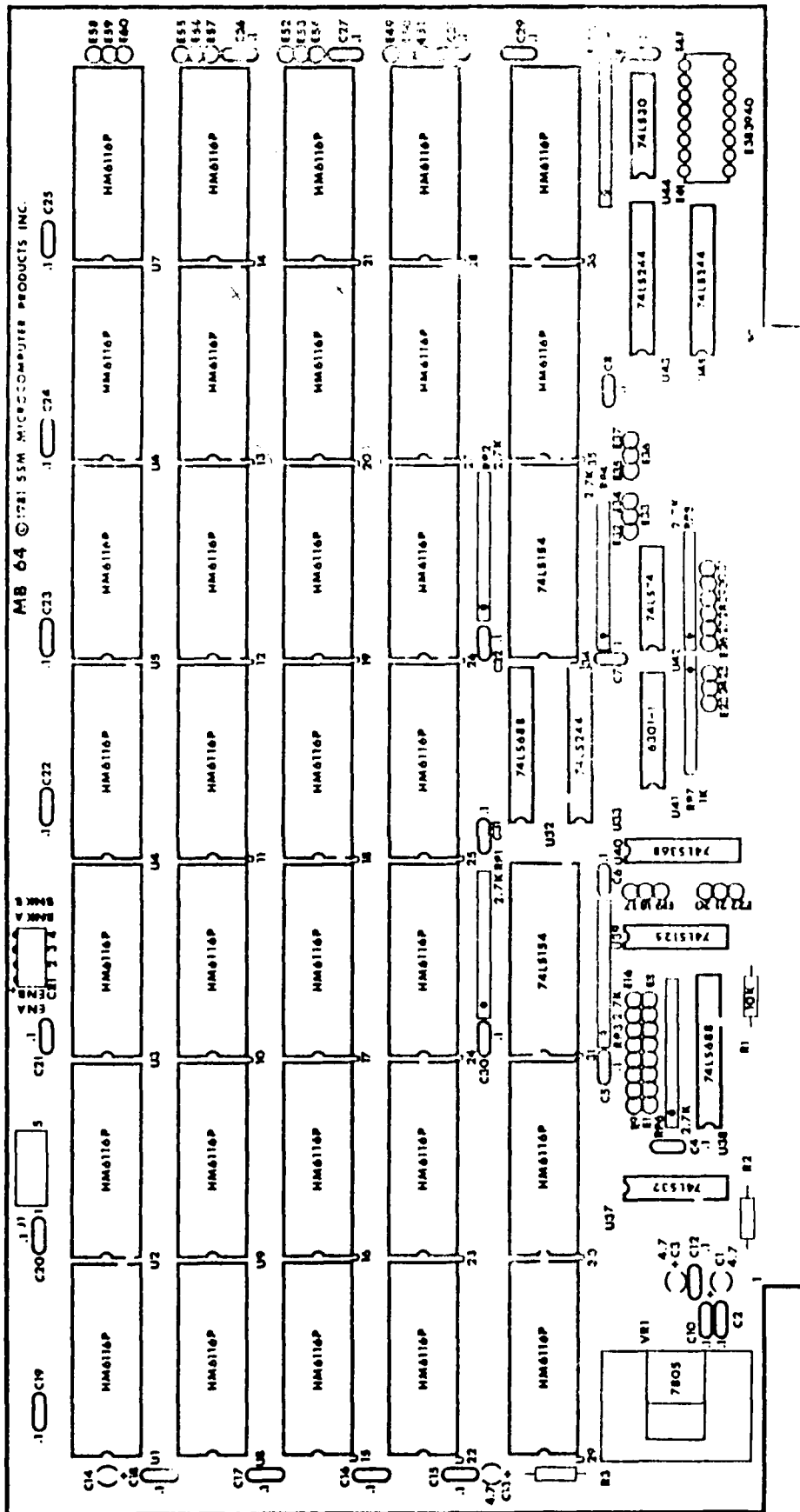
## 6.3 WARRANTY

SSM Microcomputer Products, Inc. warrants its products to be free from defects in materials and/or workmanship for a period of ninety (90) days for kits and one (1) year for factory assembled boards. In the event of malfunction or other indication of failure attributable directly to faulty workmanship and/or material, then, upon return of the product (postage paid) to SSM at 2190 Paragon Drive, San Jose, CA 95131, "Attention: Warranty Claims Department", SSM will, at its option, repair or replace the defective part or parts to restore said product to proper operating condition. All such repairs and/or replacements shall be rendered by SSM without charge for parts or labor when the product is returned within the specified period of the date of purchase. This warranty applies only to the original purchaser.

This warranty will not cover the failure of SSM products which at the discretion of SSM shall have resulted from accident, abuse, negligence, alteration, or misapplication of the product. While every effort has been made to provide clear and accurate technical information on the application of SSM products, SSM assumes no liability in any events which may arise from the use of said technical information.

This warranty is in lieu of all other warranties, expressed or implied, including warranties of mercantability and fitness for use. In no event will SSM be liable for incidental and consequential damages arising from or in any way connected with the use of its products. Some states do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

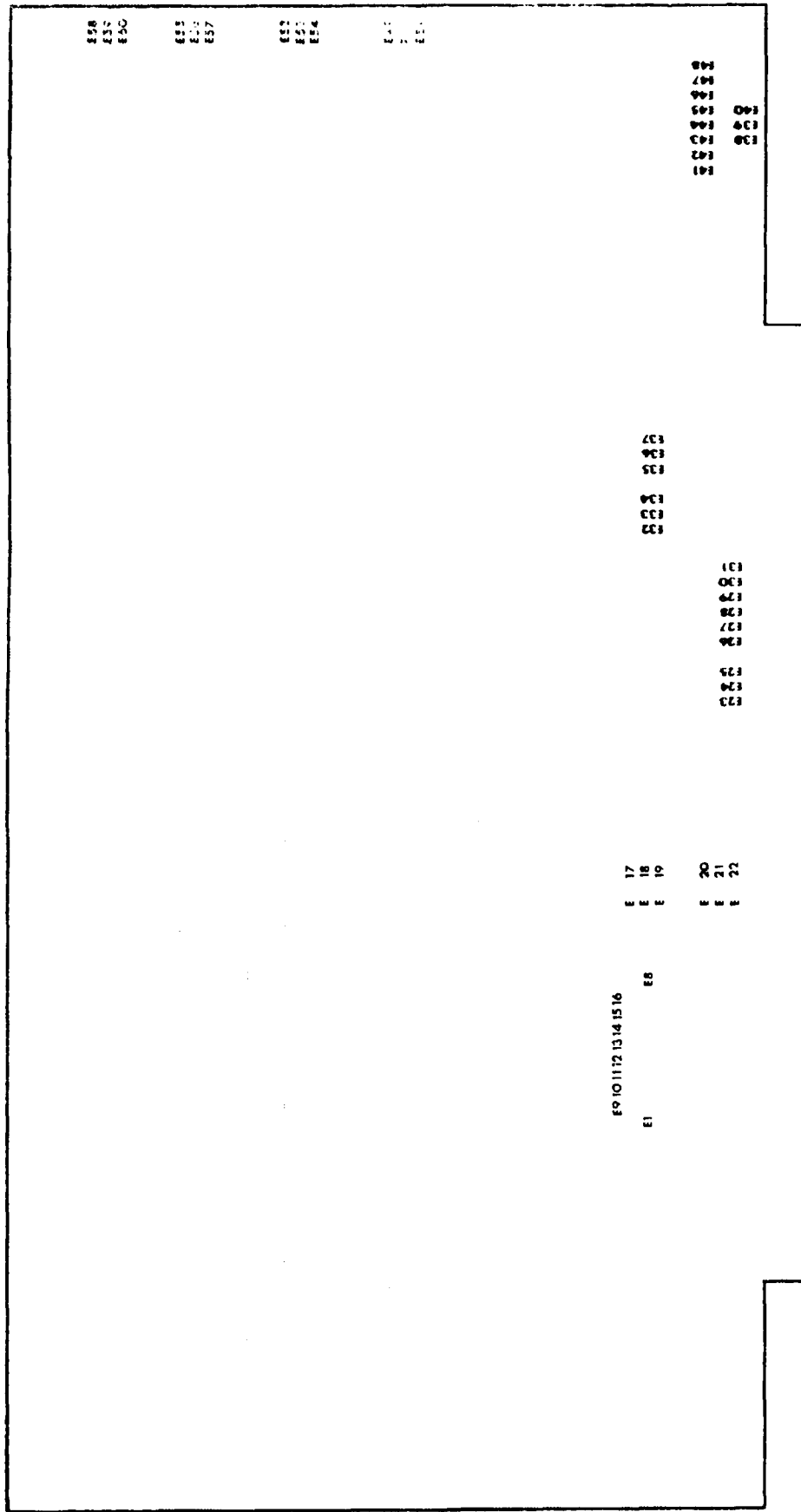
**IMPORTANT:** Proof of purchase is necessary for products returned for repair under warranty. Before returning any product, please call our Customer Service Department for a return authorization number.



J-33

ASSEMBLY DRAWING

©1981 SSM Microcomputer Products, Inc.  
All Rights Reserved



# JUMPER DRAWING

©1981 SCM Microcomputer Products, Inc.  
All Rights Reserved

A0 THRU A15 = CHIPS IN BLOCK A  
 B0 THRU B15 = CHIPS IN BLOCK B

A4 +3000	A14 +7000	A8 +4000	B2 +1000	B3 +1800	B10 +5000	B15 +7800
A3 +1800	A13 +6800	A11 +5800	B0 +0000	B4 +2000	B9 +4800	B14 +7000
A2 +1000	A15 +7800	A12 +6000	A9 +4800	B6 +3000	B8 +4000	B13 +6800
A1 +0800	A7 +3800	A6 +3000	A10 +5000	B5 +2800	B7 +3800	B12 +6000
A0 +0000	A5 +2800				B11 +5800	B1 +0800

HEX OFFSET ADDRESS FROM  
 THE START OF A 32K ADDRESS

## MEMORY MAP

© 1981 SSM Microcomputer Products, Inc.  
 All Rights Reserved

# PARTS LIST

## IC's

32	U1-U30, 35, 36	HM6116P	2K x 8 CMOS static RAM (150 ns)
2	U31, 34	74LS154	
2	U32, 38	74LS688	8-bit comparator
3	U33, 43, 45	74LS244	
1	U39	74LS125	
1	U40	74LS368	
1	U41	82S129	256 x 4 bipolar PROM (marked MB64-LE)
1	U42	74LS74	
1	U44	74LS30	
1	U37	74LS32	
1	VR1	7805	+5V voltage regulator

## RESISTORS

1	R1	10K ohm 1/4W 5% (brown, black, orange)
5	RP1, 2, 3, 4, 6	2.7K ohm 10-pin SIP resistor network
1	RP5	4.7K ohm 10-pin SIP resistor network
1	RP8	2.7K ohm 8-pin SIP resistor network
1	RP7	1K ohm 8-pin SIP resistor network

## CAPACITORS

4	C1, 3, 13, 14	4.7 uf DIP tantalum
27	2, 4-10, 12, 15-32	.1 uf monolithic capacitor

## DIODES

4	CR1-4	LED Dialight 555-2007
---	-------	-----------------------

## SOCKETS

4	14-pin sockets
3	16-pin sockets
5	20-pin sockets
34	24-pin sockets

## CONNECTORS

9	3 x 1 header strips
1	6 x 1 header strip
1	5 x 2 header strip
1	8 x 2 header strip

## HARDWARE

1	#6 hardware set
1	Small heatsink
2	Card ejectors
19	Mini-jumpers
1	MB64 PC board
1	MB64 manual
1	Warranty card
1	16-pin IC header

# HITACHI 628116P

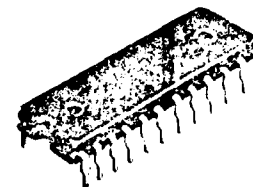
## 2048 X 8 BIT HIGH SPEED STATIC CMOS RAM

©HITACHI

### ■ FEATURES

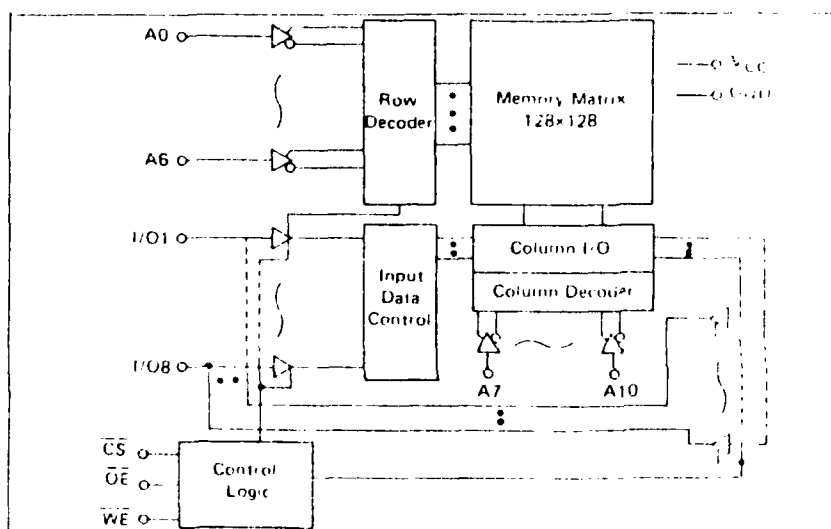
- Single 5V Supply and High Density 24 Pin Package
- High Speed Fast Access Time 120ns/150ns/200ns max
- Low Power Standby and 100 $\mu$ W typ. (Standby)
- Low Power Operation 180mW typ. (Operation)
- Completely Static RAM No clock or Timing Strobe Required
- Directly TTL Compatible All Input and Output
- Pin Out Compatible with Standard 16K EPROM/MASK ROM
- Equal Access and Cycle Time

INDUSTRIAL STANDARD  
24 pin (0.6 width)

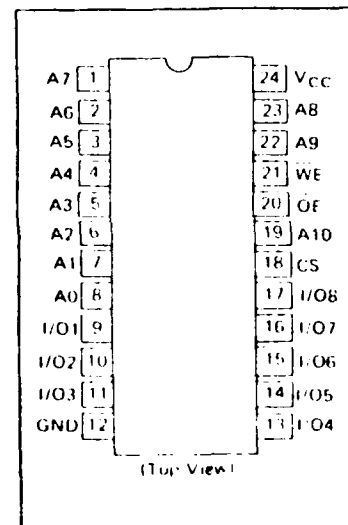


(DP-24)

### ■ FUNCTIONAL BLOCK DIAGRAM



### ■ PIN ARRANGEMENT



### ■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Rating	Unit
Voltage on Any Pin Relative to GND	$V_{IK}$	-0.5 to 7.0	V
Operating Temperature	$T_{OP}$	0 to 70	°C
Storage Temperature	$T_{STG}$	-55 to 125	°C
Temperature Under Bias	$T_{BIA}$	-10 to 85	°C
Power Dissipation	$P_T$	1.0	W

### ■ TRUTH TABLE

CS	OE	WE	Mode	$V_{CC}$ Current	I/O Pin	Ref. Cycle
H	X	X	Not Selected	$I_{DB}, I_{AK}$	High Z	
L	L	H	Read	$I_{CC}$	$D_{out}$	Read Cycle No. 1-3
L	H	L	Write	$I_{CC}$	$D_{in}$	Write Cycle No. 1
L	L	L	Write	$I_{CC}$	$D_{in}$	Write Cycle No. 2

NOTE: The specifications of this device are subject to change without notice. Please contact your nearest Hitachi's Sales Office regarding specifications.

# HM6116P SERIES

## RECOMMENDED DC OPERATING CONDITIONS (0°C ≤ Ta ≤ 70°C)

Parameter	Symbol	Min	Typ	Max	Unit	Notes
Supply Voltage	V <sub>cc</sub>	4.5	5.0	5.5	V	
	GND	0	0	0	V	
Input High (logic 1) Voltage	V <sub>ih</sub>	2.2	3.5	6.0	V	
Input Low (logic 0) Voltage	V <sub>il</sub>	*-1.0		0.8	V	*Pulse width: 50ns DC V <sub>il</sub> min: 0.3V

## DC AND OPERATING CHARACTERISTICS (0°C ≤ Ta ≤ 70°C, V<sub>cc</sub> = 5V ± 10%, GND = 0V)

Parameter	Symbol	HM6116P-2			HM6116P-3 P-4			Unit	Test Conditions	Notes
		Min	Typ	Max	Min	Typ	Max			
Input Leakage Current	I <sub>ih</sub>		1	10		1	10	μA	V <sub>cc</sub> = MAX V <sub>ih</sub> = GND to V <sub>cc</sub>	
Output Leakage Current	I <sub>ol</sub>			10			10	μA	CS = I <sub>ih</sub> or O <sub>L</sub> = V <sub>ih</sub> V <sub>cc</sub> = GND to V <sub>cc</sub>	
Operating Power Supply Current: DC	I <sub>cc</sub>		40	80		35	70	mA	CS = I <sub>ih</sub> I <sub>ol</sub> = 0mA	
Operating Power Supply Current: DC	I <sub>cc1</sub>		35			30		mA	V <sub>ih</sub> = 3.5V, V <sub>ol</sub> = 0.6V I <sub>ol</sub> = 0mA	2
Average Operating Current <sup>1</sup>	I <sub>cc2</sub>		40	80		35	70	mA	MIN cycle duty = 100%	
Standby Power Supply Current: DC	I <sub>sb</sub>		5	15		5	15	mA	CS = I <sub>ih</sub>	
Standby Power Supply Current: DC	I <sub>sb1</sub>		0.02	2		0.02	2	mA	CS = I <sub>cc</sub> = 0.2V V <sub>ih</sub> = V <sub>cc</sub> = 0.2V or V <sub>ih</sub> = 0.2V	
Output Low Voltage	V <sub>ol</sub>			*0.4			**0.4	V	*I <sub>ol</sub> = 4mA **I <sub>ol</sub> = 2.1mA	3
Output High Voltage	V <sub>oh</sub>	2.4			2.4				I <sub>ol</sub> = 1.0mA	

- NOTES 1 Typical limits are at V<sub>cc</sub> = 5.0V, Ta = +25°C and specified loading  
2 Reference Only  
3 HM6116P-2 I<sub>ol</sub> = 4.0mA, HM6116P-3 HM6116P-4 I<sub>ol</sub> = 2.1mA

## CAPACITANCE (Ta = 25°C, f = 1.0 MHz)<sup>1</sup>

Parameter	Symbol	Typ	Max	Unit	Conditions	Notes
Input Capacitance	C <sub>ix</sub>	3	5	pF	V <sub>ix</sub> = 0V	
Input-Output Capacitance	C <sub>io</sub>	5	7	pF	V <sub>cc</sub> = 0V	

NOTE 1 This parameter is sampled and not 100% tested.

## AC TEST CONDITIONS

Input pulse levels: 0.8V to 2.4V  
Input rise and fall times: 10 ns  
Input and output timing reference levels: 1.5V  
Output load: 1 TTL Gate and C<sub>L</sub> = 100pF

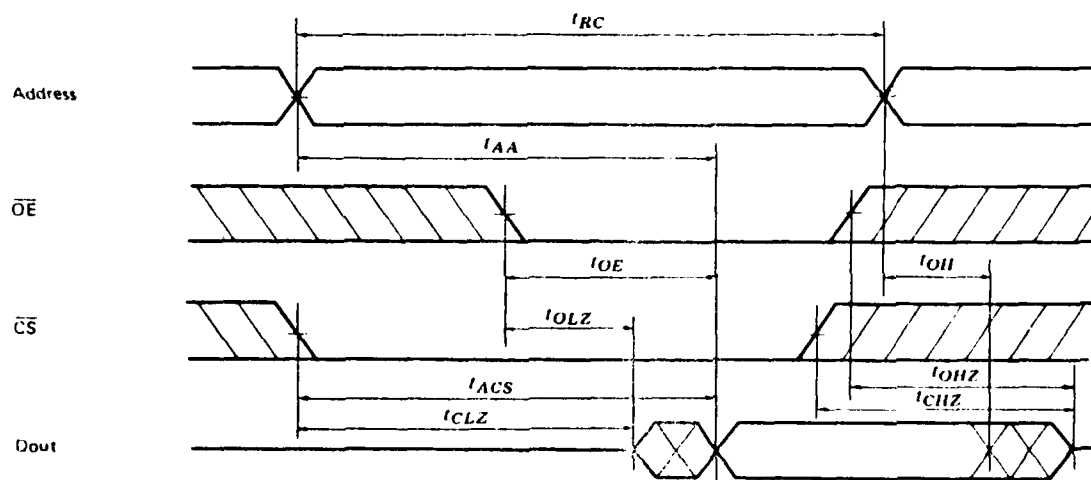
(Including scope & jig)

# AC CHARACTERISTICS (Ta = 0°C to 70°C, Vcc = 5V ± 10% unless otherwise noted)

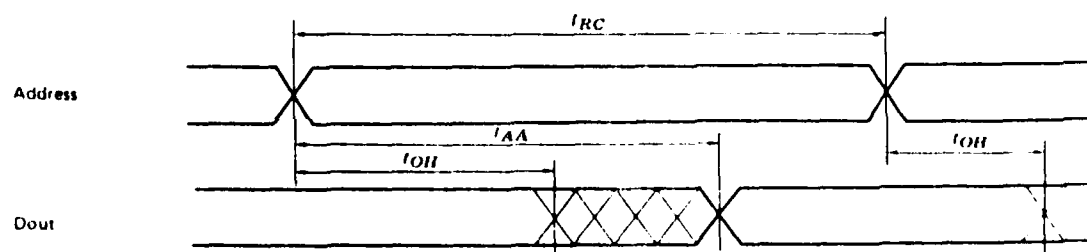
## • READ CYCLE

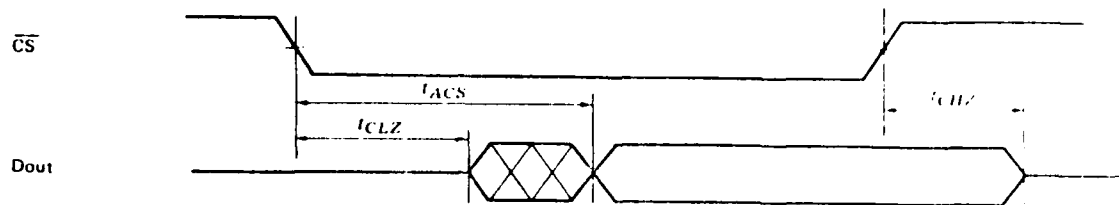
Parameter	Symbol	HM6116P-2		HM6116P-3		HM6116P-4		Unit
		Min	Max	Min	Max	Min	Max	
Read Cycle Time	$t_{RC}$	120		150		200		ns
Address Access Time	$t_{AA}$		120		150		200	ns
Chip Select Access Time	$t_{ACS}$		120		150		200	ns
Chip Selection to Output in Low Z	$t_{CLZ}$	10		15		15		ns
Output Enable to Output Valid	$t_{OE}$		80		100		120	ns
Output Enable to Output in Low Z	$t_{OLZ}$	10		15		15		ns
Chip Deselection to Output in High Z	$t_{CHZ}$	0	40	0	50	0	60	ns
Output Disable to Output in High Z	$t_{OHZ}$	0	40	0	50	0	60	ns
Output Hold from Address Change	$t_{OH}$	10		15		15		ns

## ■ TIMING WAVEFORM OF READ CYCLE NO. 1<sup>1,5</sup>



## ■ TIMING WAVEFORM OF READ CYCLE NO. 2<sup>1,2,4,5</sup>

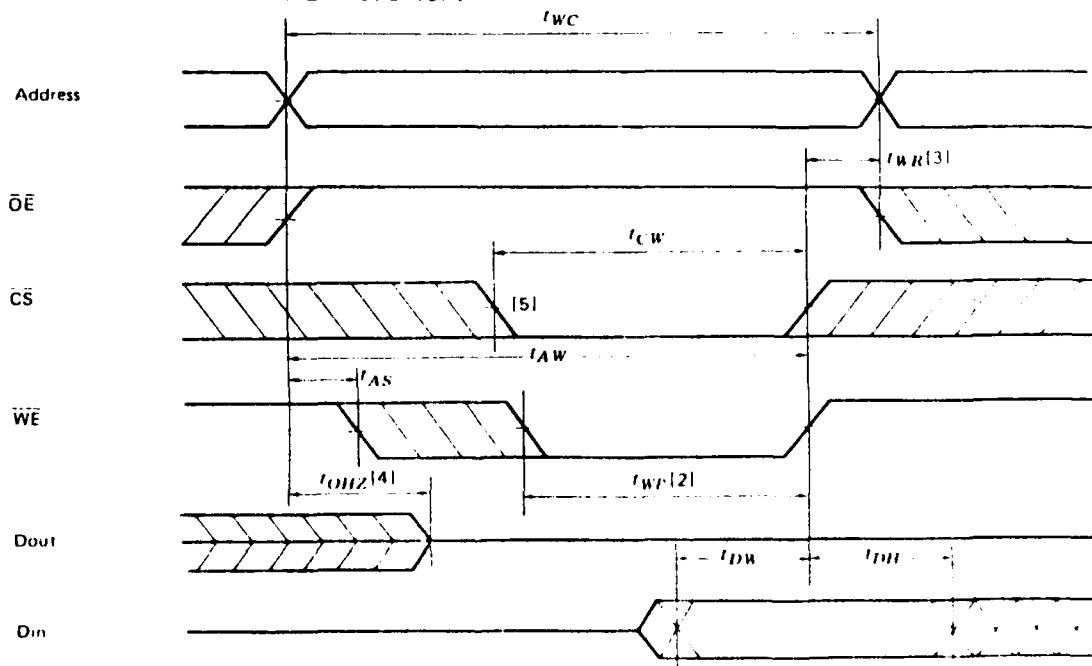


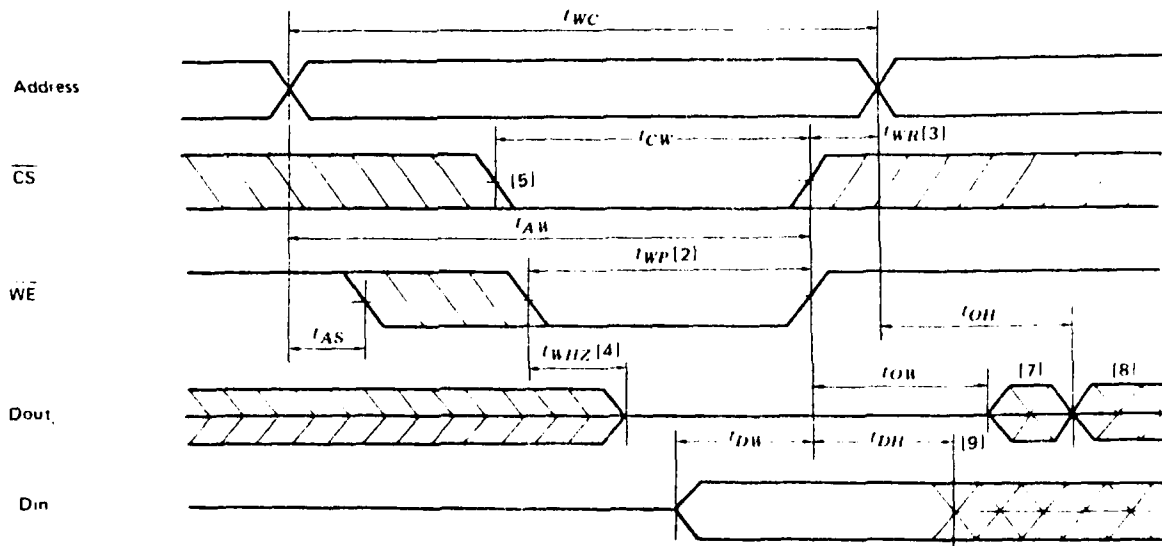
■ TIMING WAVEFORM OF READ CYCLE NO. 3<sup>1,2,4,5</sup>

- NOTES: 1. WE is High for Read Cycle.  
 2. Device is continuously selected, CS =  $V_{IL}$ .  
 3. Address Valid prior to or coincident with CS transition Low.  
 4. When CS is Low, the address input must not be in the high impedance state.

## • WRITE CYCLE

Parameter	Symbol	HM6116P-2		HM6116P-3		HM6116P-4		Unit
		Min	Max	Min	Max	Min	Max	
Write Cycle Time	$t_{WC}$	120	—	150	—	200	—	ns
Chip Selection to End of Write	$t_{CS}$	70	—	90	—	120	—	ns
Address Valid to End of Write	$t_{AV}$	105	—	120	—	140	—	ns
Address Set Up Time	$t_{AS}$	20	—	20	—	20	—	ns
Write Pulse Width	$t_{WP}$	70	—	90	—	120	—	ns
Write Recovery Time	$t_{WR}$	5	—	10	—	10	—	ns
Output Disable to Output in High Z	$t_{OHZ}$	0	40	0	50	0	60	ns
Write to Output in High Z	$t_{WHZ}$	0	50	0	60	0	60	ns
Data to Write Time Overlap	$t_{DW}$	35	—	40	—	60	—	ns
Data Hold from Write Time	$t_{DH}$	5	—	10	—	10	—	ns
Output Active from End of Write	$t_{OW}$	5	—	10	—	10	—	ns

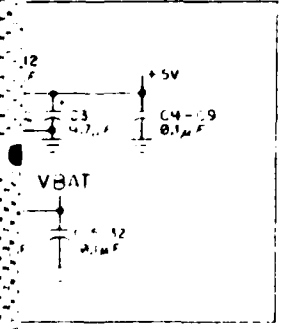
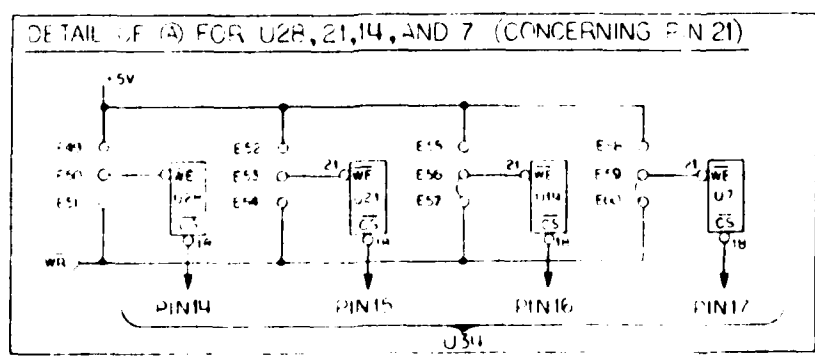
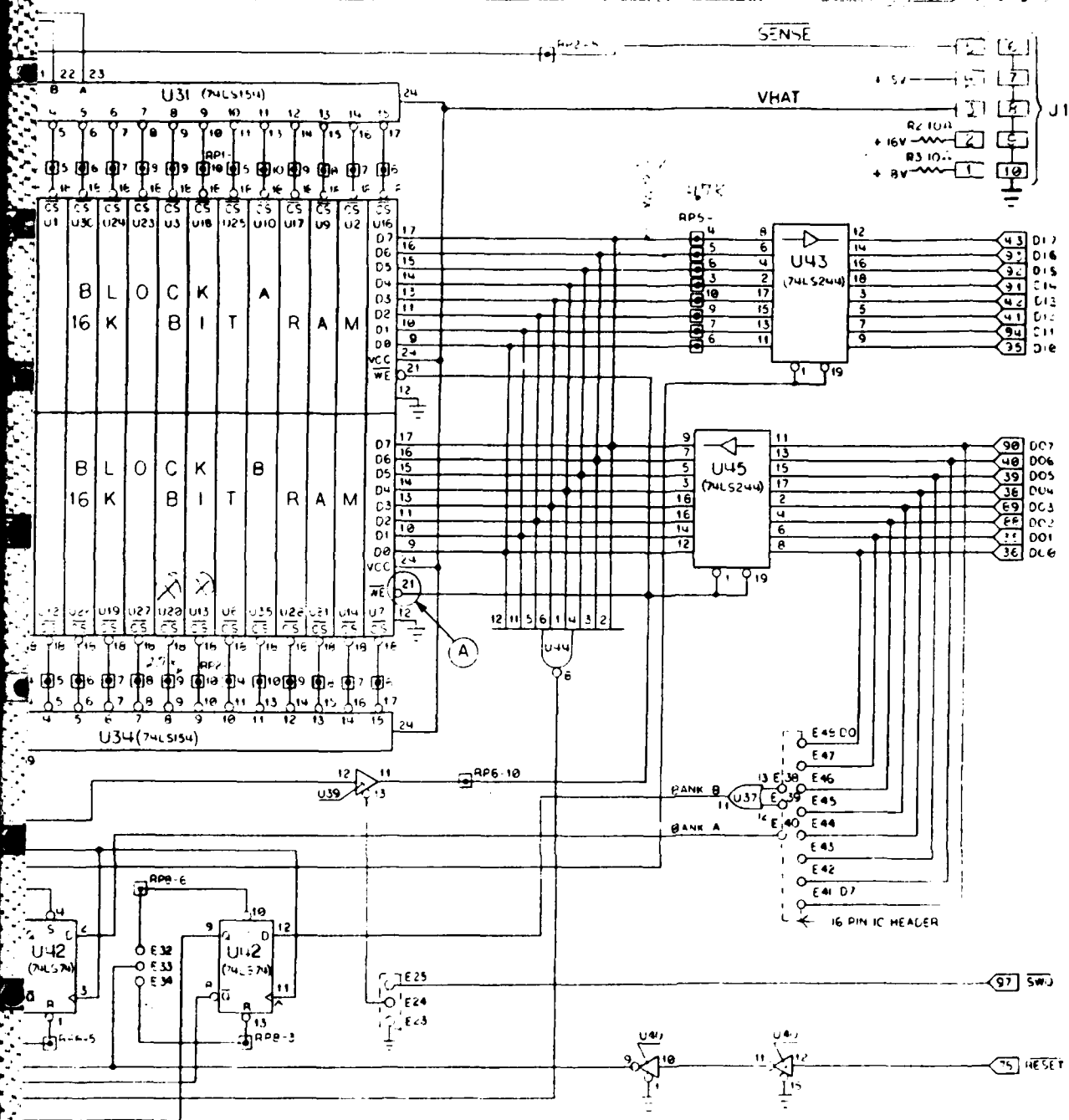
■ TIMING WAVEFORM OF WRITE CYCLE NO. 1<sup>1</sup>

■ TIMING WAVEFORM OF WRITE CYCLE NO. 2<sup>1,4</sup>

- NOTES:
- 1 WE must be high during all address transitions.
  - 2 A write occurs during the overlap ( $t_{WP}$ ) of a low CS and a low WE.
  - 3  $t_{WH}$  is measured from the earlier of CS or WE going high to the end of write cycle.
  - 4 During this period, I/O pins are in the output state so that the input signals of opposite phase to the outputs must not be applied.
  - 5 If the CS low transition occurs simultaneously with the WE low transitions or after the WE transition, output remain in a high impedance state.
  - 6 OE is continuously low ( $OE = V_{IL}$ ).
  - 7  $D_{out}$  is the same phase of write data of this write cycle.
  - 8  $D_{out}$  is the read data of next address.
  - 9 If CS is low during this period, I/O pins are in the output state. Then the data input signals of opposite phase to the outputs must not be applied to them.

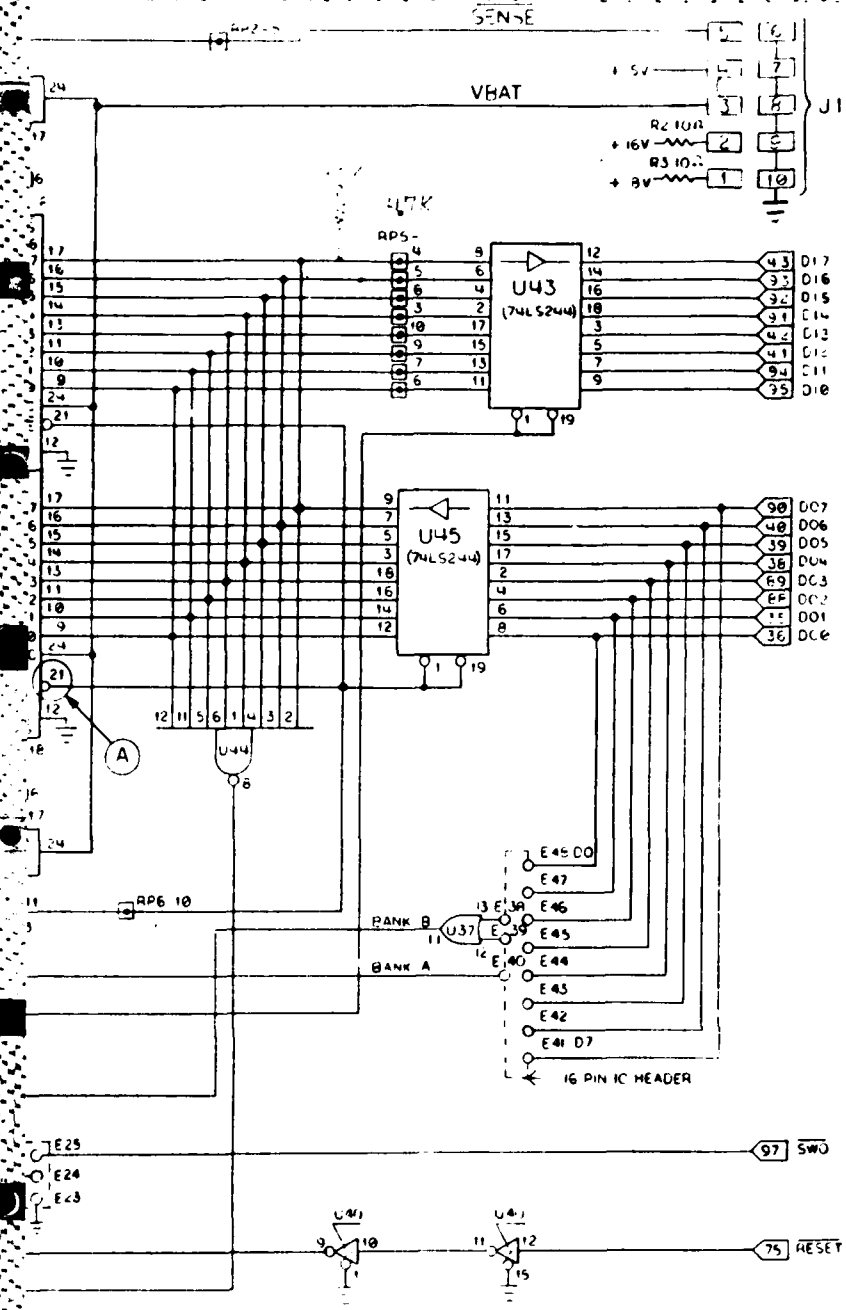


REV	
1	
A	

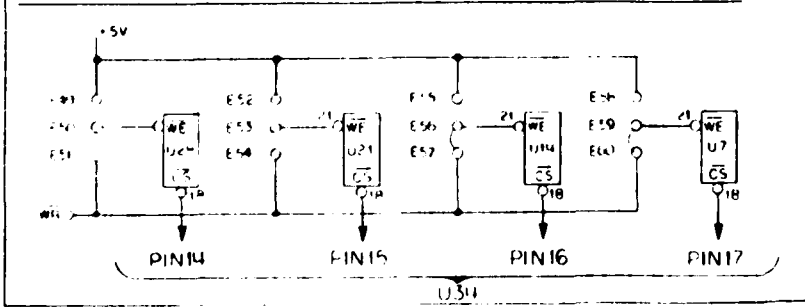


REPRODUCED AT GOVERNMENT EXPENSE

1	PROTOTYPE	10 2 51
A	PRODUCTION RELEASE	10 27 51



DETAIL OF (A) FOR U2A, 21, 14, AND 7 (CONCERNING PIN 21)



© 1981 SSM MICROCOMPUTER PRODUCTS, INC. / ALL RIGHTS RESERVED

SCALE: NONE SHEET: 1 OF 1 DATE: 10/27/51

DRAWN BY: J. A. S. I. APPROVED BY: J. A. S. I.

MB 64 DWG. # 10/27/51

# INTRODUCTION

The Imaginator is an intelligent, high efficiency, high resolution (504 by 247 pixel) graphics retrofit unit for your Heath/Zenith H/Z 19 terminal and H/Z-89 computer.

The Imaginator has its own onboard microcomputer to perform graphics processing independent of the host computer. This reduces the burden placed on the host processor and therefore improves execution speed.

A 128 character communications buffer further improves execution speed. This buffer permits the terminal and the host computer to perform their tasks asynchronously.

A graphics command may be entered by typing on the keyboard when the terminal is OFF-LINE or it may be sent via RS-232C from the host computer when the terminal is ON-LINE.

The Imaginator's transparent operation leaves all of the terminal's normal escape functions intact. The terminal's normal alphanumeric are totally independent of the Imaginator's graphics. The two displays can be overlapped on one another and may be individually altered under software control. Both alphanumeric and graphics images can be created in memory and restrained from being displayed on the screen. Once created they can be displayed instantaneously. Alternatively, the images may be displayed as they are created.

The graphics command processor (GCP) can be invoked to accept commands in either ASCII or BINARY format. ASCII mode has the advantage of easy user implementation of the graphics command language. All of the commands can be directly output by high level language programs which are executed in the host computer (e.g., PL/I, FORTRAN, PASCAL, BASIC, and of course ASSEMBLY languages). Standard, off-the-shelf, interpreters and compilers are all that are required (those languages need not have any special graphics instructions). No machine language driver programs are required.

The BINARY mode has the advantage of high efficiency. A minimum of information must be sent to specify an operation. Again, no special interpreters or compilers are required but machine language drivers are suggested (even these are not required) for efficiency.

An additional memory-mapped socket is provided for memory expansion. Up to 16K of E/P/ROM can be mounted and addressed by the GCP, or 8K of E/P/ROM and 8K of R/W RAM can be used. Custom programs can be downloaded from the host computer into this memory for fast independent execution.

## GRAPHICS INSTRUCTION SET

EnterGraphicsMode  
MoveTo (X,Y)  
PointAt (X,Y)  
LineTo (X,Y)  
AreaTo (X,Y)  
PriLineStyle (Z)  
    30 Unique styles  
SecLineStyle (Z)  
    30 Unique styles  
LineType (Z)  
    On  
    Off  
    Complement  
    Read Bit  
    Toggle to Alternate LineStyle at Boundary  
    Read Byte  
DisplayToggle (Z)  
    Enable/Disable Graphics  
    Enable/Disable Alphanumerics  
    Erase Graphics  
    or any of the eight combinations  
BringInProgram (Z<sub>0</sub>,Z<sub>1</sub>,...,Z<sub>127</sub>)  
JumpToProgram  
ExitGraphicsMode

Cleveland Codonics, Inc. reserves the right to discontinue products and to change specifications at any time without incurring any obligation to incorporate new features in products previously sold.

# HOST COMMUNICATIONS REQUIREMENTS

When operating at high baud rates, the graphics terminal will generally lag behind the host computer if asked to execute a succession of commands with long execution times (e.g., Erase, AreaTo, and LineTo commands). The Graphics Command Processor (GCP) will set the Request To Send RS-232C line false when the terminal's input communications buffer is nearly full, preventing a loss of data resulting from a buffer overflow. (The terminal's bell will tone to indicate a loss of data.) The GCP will reset the Request To Send line true when the buffer is ready to receive additional data.

Therefore, it is important that the host computer or MODEM is configured to respond to this signal. (The terminal needs no modification because it is manufactured with hardware handshaking capabilities.) A true RS-232C configuration will work fine, but often the typical RS-232C's handshaking portions are incomplete. Pin 4 of the 25-pin "D" connector on the back panel of the terminal is the Request To Send line (defined as Clear To Send at the computer end). A physical wire must connect the terminal's pin 4 with the computer's (MODEM's) pin 4.

The UARTs used in the host's RS-232C serial ports fall in two categories. Some UARTs, such as the INTEL 8251 Universal Synchronous / Asynchronous Receiver/Transmitter, respond directly to the Clear To Send signal. A high or low on the Clear To Send line with this type of UART will electronically disable or enable transmissions. This type of UART requires no further modifications.

The other type of UART has a software flag that represents the Clear To Send signal. Normally, the computer's operating system's Basic Input/Output System (BIOS) is responsible for interfacing with the serial port hardware. Generally, the BIOS will check to see if the transmitter is ready (TxRDY) before loading the UART with a character to transmit to the terminal. To add hardware handshaking, simply modify the BIOS to check the Clear To Send flag also. That is, make sure that TxRDY AND Clear To Send are both true before loading the UART with a new character to transmit.

Without this hardware handshaking, it is the programmer's responsibility to add software timing delays to prevent a buffer overflow.

Hardware handshaking will in no way detrimentally effect the operation of any of your existing programs. Software handshaking is still present when running the terminal in its standard alphanumeric mode. Assuming that the process executing in the host computer understands ctrl-S (stop transmitter) and ctrl-Q (start transmitter), it is possible to suspend graphics program output by typing a ctrl-S on the keyboard, when the terminal is on line.

The GCP supports only one directional hardware handshaking. It will send signals to control the host's serial channel transmitter, but will not respond to signals sent to the terminal's serial channel transmitter from the host.

# WELCOME

Welcome to the field of computer graphics. The human mind is the greatest known graphics processor in existence. Thoughts can be instantly conveyed by means of a picture. And in this time of information upheaval graphics is needed more than ever to enable one to assimilate it all. As a result computer graphics is one of the fastest growing disciplines in computer science.

Try typing in and executing the following demonstration programs. (We are assuming that you have access to a BASIC interpreter or compiler.)

Note that the Imaginator is assumed to be installed in a terminal that is serving as the console.

In case of error. If nothing appears to happen or something very strange happens once you have typed the RUN command give the terminal a hardware reset (right-SHIFT RESET) followed by a ctrl-C (or whatever command stops program execution in your particular version of BASIC). Type LIST and then double check the program for typing errors.

Enter and run this program first:

## DEMONSTRATION 1.

```
00010 DEFINT X,Y
00020 PRINT CHR$(27);"1"
00030 PRINT "I0,N255,D3"
00040 PRINT "M";0;125
00050 FOR X = 0 TO 500 STEP 2
00060 Y = 100*SIN(X/13.27) + 125
00070 PRINT "L";X;Y
00080 NEXT X
00090 PRINT "D6,E"
00100 STOP
```

Here's another one.

#### DEMONSTRATION 2.

```

00010 DEFINT A-Z
00020 PRINT CHR$(27);"1"
00030 PRINT "D3,I2,N255"
00040 FOR J = 1 TO 10
00050 X = 251
00060 Y = 126
00070 PRINT "P";X;Y
00080 FOR I = 0 TO 80 STEP 8
00090 X = 250-I
00100 Y = 125
00110 PRINT "L";X;Y
00120 X = 254
00130 Y = 121-I
00140 PRINT "L";X;Y
00150 X = 258 + I
00160 Y = 125
00170 PRINT "L";X;Y
00180 X = 250
00190 Y = 133 + I
00200 PRINT "L";X;Y
00210 NEXT I
00220 NEXT J
00230 PRINT "D6,E"
00240 STOP

```

Too simple? Try this one if you have some time.

This program requires the host computer to calculate over 30,000 coordinates so it takes quite a while to complete. Stop this program and relax, read the rest of the User's Guide.

#### DEMONSTRATION 3.

```

00010 DEFINT F,I,L,N,O,X,Y
00020 DIM L(302)
00030 PRINT CHR$(27);"1";"D3,N255,I0,M0,0,1000247,11"
00040 FOR I = 0 TO 301
00050 L(I) = 0
00060 NEXT I
00070 PRINT "P050023"
00080 OY = 23
00090 OX = 50
00100 FOR Y = 0 TO 100
00110 FOR X = 0 TO 300
00120 ZX = (X-150)*(X-150)/1790.5
00130 ZY = (Y-50)*(Y-50)/199
00140 Z = COS(ZX + ZY)/(SIN((ZX + ZY + .48)/82))
00150 NX = X + Y + 50
00160 NY = Y + Z + 20
00170 IF F = 1 THEN PRINT "M",NX,NY : F = 0 : GOTO 200

```

```
00180 IF NY >= L(X + 1) THEN PRINT "P",OX,OY,"L",NX,NY : GOTO 200
00190 IF NY <= L(X + 1) THEN L(X) = L(X + 1) : GOTO 210
00200 L(X) = NY
00210 OX = NX
00220 OY = NY
00230 NEXT X
00240 F = 1
00250 NEXT Y
00260 PRINT "D6,E"
00270 STOP
```

# GENERAL

## COMPUTER GRAPHICS BASICS

This is an introduction to the general concepts of computer graphics for those who may be unfamiliar with the field. Basically, a graphics terminal in its simplest form need only execute two commands: `MoveTo(X,Y)` and `LineTo(X,Y)`. A superset of commands can be formed from these two primitives.

Consider for the moment a hardcopy XY plotter. The `MoveTo(X1,Y1)` command in this case will lift the pen off the paper and move it to the absolute coordinate (X<sub>1</sub>,Y<sub>1</sub>). The `LineTo(X2,Y2)` command will drop the pen onto the paper and move it in a straight line to the absolute coordinate (X<sub>2</sub>,Y<sub>2</sub>) (i.e., it would draw a line segment from (X<sub>1</sub>,Y<sub>1</sub>) to (X<sub>2</sub>,Y<sub>2</sub>)).

In a CRT style graphics terminal the commands would be executed in a similar manner. The `MoveTo(X1,Y1)` command will move a virtual pointer to the absolute screen coordinate (X<sub>1</sub>,Y<sub>1</sub>). Nothing is written on the screen. The `LineTo(X2,Y2)` command writes a straight line on the screen from the absolute coordinate (X<sub>1</sub>,Y<sub>1</sub>) to the absolute coordinate (X<sub>2</sub>,Y<sub>2</sub>) by turning on the appropriate pixels (picture elements). Almost any geometrical shape can be created by a sequence of `MoveTo` and `LineTo` commands (e.g., a circle can be approximated by a many sided polygon). Several other primitive utility commands are convenient, such as some means to erase the screen and a command to reinitialize the graphics terminal. To take some of the burden from the applications programmer, this primitive instruction set is usually expanded.

## IMAGINATOR SPECIFICS

The graphics screen memory is composed of 131072 bit arranged in a 512 by 256 array (although only 504

by 247 are user accessible and displayed). The positive X axis (horizontal axis) originates at the left of the screen and terminates at the right. The positive Y axis (vertical axis) originates at the bottom of the screen and ends at the top. Therefore, the origin (0,0) is located at the lower left of the screen. Since the alphanumeric screen is 80 characters wide and the graphics screen is 63 characters wide, the graphics screen's left starts at the alphanumeric's 9th character position.

To view the entire graphics screen, enable the 25th line, `ESC x 1` (`ESC [ 1 h` if in ANSI mode).

When the terminal is reset, either when powered up, a keyboard reset, **right SHIFT-RESET**, or a software reset `ESC z` (`ESC [ z` if in ANSI mode) the terminal will perform as though it were unmodified. It will execute all of the escape functions it did before the addition of the Imaginator—the functional existence of the Imaginator is transparent to the user. (At this time the graphic's video RAM will be cleared, and the line type will be ON; the primary line style will be solid, the secondary line style will be blank, and the virtual pointer will be assigned as (0,0).)

To invoke the graphics command processor (GCP), an "EnterGraphicMode" escape sequence is required. (When graphics or "EnterGraphics Mode" is referred to in this manual it should be connoted as a reference to the capabilities of the Imaginator, not the 33 special symbols stored in the terminal's character generator.)

The GCP can be invoked to accept commands in either ASCII mode or as seven bit binary words (BINARY mode). Both forms of each command will be accompanied by a functional description.

A command may be entered either by typing on the keyboard when the terminal is OFF LINE or it may be sent via RS-232C from the host computer when the terminal is ON LINE.

There is no good way to abort a command midway (e.g., delete and backspace won't erase a command). Obviously, a keyboard reset **right SHIFT-RESET** is one way to clear a half-created command, but is rather drastic. The GCP expects to receive commands and data in certain fixed sequences; once a command sequence is started it must be completed.

# COMMAND FORM AND FUNCTION, ASCII

## ASCII COMMAND FORMATS

A complete description of the form and function of each command follows.

Upper case characters A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P represent commands (some of these are unassigned).

**X** represents the absolute horizontal coordinate. It must be an integer between 0 and 999 inclusively, although it will be truncated to 503 if greater than 503.

**Y** represents the absolute vertical coordinate. It must be an integer between 0 and 999 inclusively, although it will be truncated to 246 if greater than 246.

**Z** represents an operand. It must be an integer between 0 and 999 inclusively.

[opt. delim] represents an optional delimiter. A delimiter here is not required but may be included. If included, it may be any number of ASCII characters except the characters 0,1,2,3,4,5,6,7,8,9.

[delim] represents a delimiter. A delimiter here is mandatory unless three consecutive numerals precede it (a delimiter is automatically assumed after a three digit number; additional delimiters are optional). The delimiter may be any ASCII character except 0,1,2,3,4,5,6,7,8,9.

It will be assumed in the remainder of this manual that the language BASIC is understood by the reader. However, only the most rudimentary of BASIC commands will be used to prevent undue confusion to a novice.

The following examples illustrate a typical command format:

A PointAt command: **P** [opt. delim] **X** [delim] **Y** [delim] may be created in BASIC as:

**PRINT "P";X;Y**      The space will serve as the delimiter.

or

**PRINT "P",X,Y**      The tab will serve as the delimiter, (note that in some BASICs a tab may be represented as a series of spaces. This format would then be inefficient.)

or

**PRINT "P"**  
**PRINT X**  
**PRINT Y**      The carriage return/line feed will serve as the delimiter.

or

If X and Y are constants such as X=25 and Y=39

**PRINT "P";25;39**      The space will again serve as the delimiter.

or

**PRINT "P025039"**      The leading zeros create three digit numbers so the delimiter is automatically inserted.

## EnterGraphicsMode, ASCII

**Command form:** ESC 1

**Command function:**

This command signals the GCP to interpret all future information as graphics command/data. No graphics attributes are reinitialized. Commands and data will now be assumed to consist of the ASCII characters A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P and 0,1,2,3,4,5,6,7,8,9 respectively. ASCII mode has the advantage of easy user implementation of the graphics command language. All of the commands can be directly output by high level language programs which are executed in the host computer. No machine language driver programs are required. The ASCII mode has the disadvantage of inefficiency. On the average, twice as many characters must be sent to the terminal than in binary mode to perform the same operation. The disadvantage would be most evident when communications speed, rather than vector drawing speed or host processor speed, is the effective bottleneck.

**EXAMPLE:**           10 PRINT CHR\$(27);"1"

## MoveTo (X,Y), ASCII

**Command form:** M [opt. delim] X [delim] Y [delim]

**Command function:**

The virtual pointer is assigned the absolute coordinate (X,Y). Nothing is written to the screen nor can it be interrogated.

**EXAMPLE:**

```
10 DEFINT X,Y
.
.
.
20 X = 25
30 Y = 210
40 PRINT "M";X;Y
```

## PointAt (X,Y), ASCII

**Command form:** P [opt. delim] X [delim] Y [delim]

**Command function:**

The virtual pointer is assigned the absolute coordinate (X,Y). The Pattern byte (see the LineStyle commands) is rotated one position; if the carry contains a 0, the command is treated as a MoveTo, command. If the carry contains a 1, the pixel is interacted with according to the pending line type (see LineType command).

**EXAMPLE:**

```
10 DEFINT X,Y
.
.
.
20 X = 25
30 Y = 210
40 PRINT "P";X,Y
```

## LineTo (X,Y), ASCII

**Command form:** L [opt. delim] X [delim] Y [delim]

**Command function:**

A line is drawn from, but not including, the virtual pointer's currently assigned absolute coordinate to the absolute coordinate (X,Y). The line drawn is subject to the current line style and line type attributes. This command will emulate a MoveTo command if the line style is 00000000 (execution time will be considerably longer however). At the completion of this command, the virtual pointer is assigned the absolute coordinate (X,Y).

**EXAMPLE:**

```
10 DEFINT X,Y
.
.
.
20 X = 25
30 Y = 210
40 PRINT "L";X;Y
```

## AreaTo (X,Y), ASCII

**Command form:** A [opt. delim] X [delim] Y [delim]

### Command function:

The area inside a regular rectangle is filled. The rectangle is defined as having the virtual pointer's currently assigned absolute address as one vertex and the absolute coordinate (X,Y) as the diagonally opposite vertex. Starting at, but not including, the virtual pointer's currently assigned absolute coordinate, a horizontal line is drawn to the opposite side of the rectangle. When possible, a second line starting at the original side of the rectangle is drawn adjacent to the first line (a rectangle with a height of 1 will only accept one line). This procedure is repeated until the rectangle is filled. The line drawn is subject to the current line style and line type attributes. This command will behave as a MoveTo command if the line style is 00000000 (execution time will be considerably longer however). At the completion of this command the virtual pointer is assigned the absolute coordinate (X,Y).

### EXAMPLE:

```
10 DEFINT X,Y
.
.
.
20 X = 25
30 Y = 210
40 PRINT "A";X;Y
```

## PriLineStyle (Z), ASCII

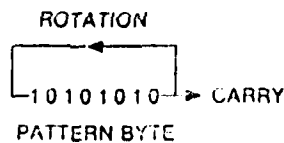
**Command form:** N [opt. delim] Z [delim]

**Where:** Z is a number between 0 and 399 inclusively. This number is converted to binary format whose least significant 8 bits are used to define the Primary Pattern.

### Command function:

This command permits dashed or dotted lines to be automatically generated by the GCP.

Preceding any write to the graphics display, the pending Pattern byte is rotated one position. The least significant bit is rotated into the carry and is used to determine whether screen interaction is permitted or not. A logical 1 in the Pattern represents permission to interact with the pixel; a 0 disables interaction. The pending Pattern byte is then updated with the new rotated pattern. The least significant bit is the first to be tested to determine if interaction should occur. Therefore, the eight bit line style pattern is repetitively traced to the screen when drawing a line.



The LineStyle and LineType commands are totally independent of one another. The line style will equally effect any line type attribute (except READ BIT and READ BYTE). For instance, a line drawn with a 10101010 line style and a complement line type will complement every other pixel.

When short line segments are used to construct long lines (e.g., curves), they should be sent in a consecutive order. There is no guarantee that a line segment patched into the middle of an existing line will have a perfectly matched line style sequence. Of course, it is possible to reset the sequence by executing another LineStyle command.

The pending line style pattern is always reset to Primary when entering any graphics command.

→ Any portion of the graphics display may be selectively erased by executing an AreaTo command with a line style of 11111111 and an OFF line type.

**EXAMPLE:** 10 PRINT "N255"

## SecLineStyle (Z), ASCII

**Command form:** O [opt. delim] Z [delim]

**Where:** Z is a number between 0 and 999 inclusively. This number is converted to binary format whose least significant 8 bits are used to define the Secondary Pattern.

**Command function:**

Identical to PriLineStyle (Z), ASCII

**EXAMPLE:**           10 PRINT "O170"

## LineType (Z), ASCII

Command form: I [opt. delim] Z [delim]

Where:	Z	PIXEL ACTION
	0	ON
	1	OFF
	2	COMPLEMENT
	3	READ BIT
	4	TOGGLE TO ALTERNATE LIFESTYLE AT BOUNDARY
	5	READ BYTE

### Command function:

This command sets the type of line to be drawn. (Note, that a point is considered a short line and an area is considered a long line). Consider each pixel of the line individually for now.

The different line types are explained below.

**ON**—the pixel is turned on.

**OFF**—the pixel is turned off (i.e., erased).

**COMPLEMENT**—the pixel is complemented (i.e., the pixel is turned on if it was off and it is turned off if it was on).

**READ BIT**—The pixel is interrogated to determine whether it is on or off but is not otherwise effected. An ASCII 0 or 1 followed by a carriage return is transmitted to the host computer for a pixel that is respectively off or on.

This line type has some special restrictions.

This line type can only be used in conjunction with a PointAt command. LineTo and AreaTo commands will imitate a MoveTo command.

Note that if the terminal is OFF LINE this attribute will perform no function except that the PointAt, LineTo, or AreaTo command will act as a MoveTo command.

The line style will act as if it were set to solid (1111111) regardless of its actual value. (See LineStyle command). This is to prevent the host computer from getting trapped in an eternal wait loop for a terminal response if the line style contains a 0.

The process executing in the host computer that is responsible for reading the data sent by the terminal must be fast enough to keep pace. The terminal will transmit the data as fast as the baud rate selected will permit.

It is important that the host computer does not echo the terminal response (0 or 1 followed by a carriage return) back to the terminal. An echoed response will be treated by the GCP as command/data information. (This is really only true if the GCP is in BINARY mode, because in ASCII mode the 0 or 1 will be received when the GCP is expecting an opcode (A—P) and will therefore be assumed to be a delimiter.) See the Examples section of this manual to see how this can be implemented.

**TOGGLE TO ALTERNATE LIFESTYLE AT BOUNDARY**—This line type is a very simple, and therefore limited, algorithm that may be used for filling irregular polygons.

As the line is scanned, each pixel is interrogated in turn to determine whether it is on or off. If it is off it is written to according to the pending line style. A single on pixel will be left untouched, but the current line style pattern is exchanged with the alternate Pattern. For instance, if the line style is currently loaded with the Primary Pattern it will be reloaded with the Secondary Pattern, or if the Lifestyle is currently loaded with the Secondary Pattern it will be reloaded with the Primary Pattern. If two or more adjacent pixels are on they will be left untouched and line style pattern will NOT be exchanged. At the completion of the LineTo or AreaTo command the line style is reloaded with the Primary Pattern.

**READ BYTE**—The display byte is read and converted from binary to hexadecimal. The ASCII representation of this hexadecimal number is transmitted to the host computer. Display bytes are defined as 8 consecutive horizontal pixel locations. The beginning of a display byte is (X,Y) where X is 0,8,16,...,400 and Y is any integer between 0 and 246, inclusively. Each display byte is redundantly addressed by 8 coordinates. For example, to access the display byte beginning at (0,0) any of the following coordinates could be used: (0,0), (1,0), (2,0), (3,0), (4,0), (5,0), (6,0), or (7,0). The pixel at the beginning of the display byte is the least significant and the pixel at the beginning + 8 is the most significant. Notice that this means that, visually, a pattern on the screen will appear in reverse significance with respect to its hexadecimal representation.

Leading zeros are transmitted (not suppressed).

This line type has some special restrictions.

This line type can only be used in conjunction with a PointAt command. LineTo and AreaTo commands will imitate a MoveTo command.

Note that if the terminal is OFF LINE this attribute will perform no function except that the PointAt, LineTo or AreaTo command will act as a MoveTo command.

The line style will act as if it were set to solid (1111111) regardless of its actual value. (See LineStyle command) This is to prevent the host computer from getting trapped in an eternal wait loop for a terminal response if the line style contains a 0.

The process executing in the host computer that is responsible for reading the data sent by the terminal must be fast enough to keep pace. The terminal will transmit the data as fast as the baud rate selected will permit.

It is important that the host computer does not echo the terminal response (00 to FF followed by a carriage return) back to the terminal. An echoed response will be treated by the GCP as command/data information. See the Examples section of this manual to see how this can be implemented.

## DisplayToggle (Z), ASCII

Command form: D [opt. delim] Z [delim]

Where:	Z	ENABLE ALPHA	ENABLE GRAPHICS	ERASE GRAPHICS
	10	NO	NO	NO
	11	NO	NO	YES
	12	NO	YES	NO
	13	NO	YES	YES
	14	YES	NO	NO
	15	YES	NO	YES
	16	YES	YES	NO
	17	YES	YES	YES

### Command function:

This command has two distinct functions. One function is to permit the user to block or not block the display of alphanumeric or graphics information to the entire screen. The other function of this command is to erase the entire graphics display memory. This command stays in effect even after executing an ExitGraphicsMode command.

**EXAMPLE:** 10 PRINT "D3"

This command would disable alphanumerics, enable graphics and erase the previous image.

## BringInProgram (Z0), (Z1), ... ,(Z127), ASCII

**Command form:** B [opt. delim] Z0 [opt. delim] Z1 [opt. delim] ... ,Z127 [opt. delim]

Where: [opt. delim] in this case is any ASCII character except 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

**AND**

Where: Z is a 2-digit hexadecimal number between 00 and FF, inclusively. A leading zero must be present if a single digit number (i.e., 03 not 3). However, do not insert a leading zero in front of a two digit number (i.e., FF not 0FF).

**Command function:**

This command loads 128 bytes of data (Z0-Z127) into the expansion R/W RAM U9B. The data is converted from hexadecimal to binary format prior to loading into R/W RAM. Z0 is loaded into memory at address C001H, Z1 is loaded into memory at C002H, etc. After the 128th byte is loaded, control is returned to the GCP for the next command.

This command is only useful if R/W RAM is mounted at U9B. Beware that once a BringInProgram command is initiated, the GCP will expect at least 256 characters before accepting new commands (this is true regardless of whether R/W RAM is present at U9B or not).

**EXAMPLE:**

```

10 PRINT "B"
20 PRINT "00"
30 PRINT "00"
40 PRINT "00"
50 PRINT "C3"
60 PRINT "04"
70 PRINT "C0"
.
.
.
1290 PRINT "00"
```

This example of data entry is correct with regard to format but is quite inflexible and therefore not advocated as a good programming technique.

## JumpToProgram, ASCII

**Command form: J**

**Command function:**

This command transfers control from the GCP to the program residing in U9B. Transfer is accomplished by a JMP (JUMP) to address C004H. Control may be given back to the GCP by a RET (RETURN) statement.

Before the transfer is made, a test pattern is written to location C000H and then read back. The pattern must match or no transfer is permitted and control is returned to the GCP. Therefore, physical memory must be mounted at U9B and it must be valid at C000H. This prevents inadvertently jumping to a nonexistent program, resulting in a runaway processor.

•  
**EXAMPLE:**           10 PRINT "J"

## ExitGraphicsMode, ASCII

Command form: E

Command function:

This command instructs the GCP to release control back to normal alphanumeric processing. All previously set graphics attributes will remain valid (i.e., no attributes revert back to default or reset values).

EXAMPLE:           10 PRINT "E"

## Appendix E

### The CASM Assembly-language-to-CRL-Format Preprocessor For BDS C v1.50

The only means previously provided to BDS C users for creating relocatable object modules (CRL files) from assembly language programs was a painfully complex macro package (CMAC.LIB) that only operated in conjunction with Digital Research's macro assembler (MAC.COM). This was especially bad because MAC, if not already owned, cost about as much as the entire BDS C package to purchase. This document describes the program "CASM", supplied to eliminate the need for "MAC". CASM is a preprocessor that takes, as input, an assembly language source file of type ".CSM" (mnemonic for C aSseMbly language) in a format much closer to "vanilla" assembly language than the bizarre craziness of CMAC.LIB, and writes out an ".ASM" file which may then be assembled by the standard, ubiquitous CP/M assembler (ASM.COM). CASM automatically recognizes which assembly language instructions require relocation parameters, and inserts the appropriate pseudo-operations and extra opcodes into the resulting ".ASM" file so it properly assembles directly into CRL format. In addition, some rudimentary logic checks are performed: doubly-defined and/or undefined labels are detected and reported, and similarly-named labels in different functions are ALLOWED and converted into unique names so ASM won't complain.

#### E.1 Creating CASM.COM

CASM is supplied in source form only on the BDS C distribution disk. Before compiling CASM.C to make an executable version, customize the beginning of the file by setting the default library drive and/or user area definitions to conform to your system configuration. Instructions for compilation and linkage of CASM are given in the comments at the head of the file.

## E.2 Command Line Options

- c                    Enables comment retention on both input and output. By default, CASM strips off all comments from the input file when reading it in, and does not put any comments into the assembly code added to form the final ASM file. If -c is specified, the original comments are preserved and CASM adds its own comments to new sections of code.
- f                    Flags old CMAC.LIB macro library operators, to help users convert old assembly language source files to the CSM format.
- o name            Calls the output file name.ASM. Normally, the output file is named by tacking an .ASM extension onto the filename of the CSM input file.

The files making up the CASM package are as follows:

CASM.C	Source file for CASM program
CASM.SUB	Submit file for performing the entire conversion of a CSM file into CRL format
ASM.COM (or MAC.COM)	Standard CP/M utility, for assembling the output of CASM.
DDT.COM (or SID.COM)	Standard CP/M utility, for converting the HEX output of the assembler into binary CRL format.

The pseudo-operations that CASM recognizes as special control commands within a .CSM file are as follows:

**FUNCTION <name>** Each function must begin with a FUNCTION pseudo-op, where <name> is the name that will be used for the function in the .CRL file directory. No other information should appear on this line. Note that there is no need to specify a complete

list of contained functions at the start of a .CSM file, as was the case with the old CMAC.LIB method of CRL file generation.

**EXTERNAL <list>** If a function calls other C or assembly-coded functions, an **EXTERNAL** pseudo-op naming these other functions must follow immediately after the **FUNCTION** op. One or more names may appear in the list, and the list may be spread over as many **EXTERNAL** lines as necessary. Only function names may appear in **EXTERNAL** lines; data names (such as "external" variables defined in C programs) cannot be placed in "external" statements.

**ENDFUNC**

**ENDFUNCTION** This op (both forms are equivalent) must appear after the end of the code for a particular function. The name of the function need not be given as an operand. The three pseudo-ops just listed are the **ONLY** pseudo-ops that need to appear among the assembly language instructions of a ".CSM" file, and at no time do the assembly instruction themselves need to be altered for relocation, as was the case with CMAC.LIB.

**INCLUDE <filename>**

**INCLUDE "filename"** This op causes the named file to be inserted at the current line of the output file. If the filename is enclosed in angle brackets (i.e., <filename>) then a default CP/M logical drive is presumed to contain the named file (the specific default for your system may be customized by changing the appropriate #define in CASM.C). If the name is enclosed in quotes, then the current drive is searched. Note that you'll usually want to include the file BDS.LIB at the start of your .CSM file, so that names of routines in the run-time package are recognized by CASM and not interpreted as undefined local forward references...since CASM is a one-pass preprocessor, that would cause it to generate undesired relocation parameters for instructions having run-time package routine names as operands. Note that the pseudo-op **MACLIB** is equivalent to **INCLUDE** and may be used instead.

The format for a ".CSM" file is as follows:

```

INCLUDE      bds.lib

FUNCTION     function1
[ EXTERNAL   needed_func1 [,needed_func2] [...] ]
code for function1
ENDFUNC

FUNCTION     function2
[ EXTERNAL   needed_func1 [,needed_func2] [...] ]
code for function2
ENDFUNC

```

Additional notes and bugs:

1. If a label appears on an instruction, it must begin in column 1 of the line. If a label does not begin in column 1, CASM will not recognize it as a label and relocation will not be handled correctly.
2. Forward references to EQUated symbols in executable instructions are not allowed, although forward references to relocatable symbols are OK. The reason for this is that CASM is a one-pass preprocessor, and any time a previously unknown symbol is encountered in an instruction, CASM assumes that symbol is relocatable and generates a relocation parameter for the instruction.
3. INCLUDE (and MACLIB) only work for one level of inclusion.
4. When a relocatable value needs to be specified in a dw op, then it must be the only value given in that particular DW statement, or else relocation will not be properly handled. In other words, only one 16-bit relocatable item is allowed per dw statement.
5. Characters used in symbol names should be restricted to alphanumeric characters; the dollar sign (\$) is also allowed, but might lead to a conflict with labels generated by CASM.
6. The .HEX file produced by ASM after assembling the output of CASM cannot be converted into a binary file by using the CP/M LOAD command; instead, DDT or SID must be used to read the file into memory, and then the CP/M SAVE command must be issued to save the file as a .CRL file. CASM inserts a line into the ASM file ending in the character sequence "!",

specifically so that the line will be flagged as an error... the user may then look at the value printed out at the left margin to see exactly how many 256-byte blocks need to be SAVED after using DDT or SID to get the file into memory. The reason that LOAD cannot be used is that CASM puts out the code to generate the CRL File directory at the end of the ASM file, using the "ORG" pseudo-op to set the location counter back to the base of the TPA. The LOAD command aborts with the cryptic message "INVERTED LOAD ADDRESS" when out-of-sequence data of this nature is encountered. Rather than having CASM write out the directory information into a new file and then append the entire previous output onto the end of this new directory file, I decided to require the user to enter a SAVE command.

7. The CASM.SUB submit file may be used to perform the entire procedure of converting a ".CSM" file to a ".CRL" file, except for entering the final SAVE command. For a file named "FOO.CSM", just say:

```
submit casm foo
```

and enter the "SAVE" command just the way it instructs you to when processing is complete.

## Appendix I

### A Long Integer Package for BDS-C

Rob Shostak  
August, 1982

#### I.1 Introduction

This package adds long (32-bit) signed integer capability to BDS C much in the same spirit as Bob Mathias's floating point package. Addition, subtraction, multiplication, division, and modulus routines are provided as well as comparison, assignment, and various kinds of conversion.

Each long integer is stored as an array of four characters. A long integer *x* is thus declared by:

```
char x[4];
```

The internal representation is two's complement form, with the sign (most significant) byte as the first byte of the array. For most purposes, however, you needn't be concerned with the internal representation.

Most of the routines that operate on longs take three arguments, the first of which points to where the result is to be stored, and the other two of which give the operands. For example, given longs *x*, *y*, and *z* (all declared as `char[4]`),

```
ladd(z,x,y)
```

computes the sum of *x* and *y* and stores it into *z*, which is returned as the value of the call. Note that the result argument may legitimately be the same as one (or both) of the operand arguments (for instance, `ladd(x,x,x)` does "the right thing").

The package is written partly in C and partly (for speed and compactness) in 8080 assembly language. To use it, simply link LONG.CRL into your program. A description is given below for each routine.

```
itol(l,i)
char l[4];
int i;
```

Stores the long representation of the 16-bit integer i into l, and returns l.

```
atol(l,s)
char l[4];
char *s;
```

Stores the long representation of the Ascii string s into l, and returns l. The general form of s is a string of decimal digits, possibly preceded by a minus sign, and terminated by any non-digit.

```
ltoa(s,l)
char *s;
char l[4];
```

Stores the Ascii representation of long l into string s, and returns s. The representation consists of a null-terminated string of Ascii digits preceded by a minus sign if l is negative. s must be large enough to receive the conversion.

```
ladd(r,op1,op2)
char r[4];
char op1[4], op2[4];
```

Stores the sum of longs op1 and op2 into r, and returns r. op1 or op2 may be used for r.

```
lsub(r,op1,op2)
char r[4];
char op1[4],op2[4];
```

Similar to ladd, but computes op1 - op2.

```
lmul(r,op1,op2)
char r[4];
char op1[4],op2[4];
```

Similar to ladd, but computes op1 \* op2.

```
ldiv(r, op1, op2)
char r[4];
char op1[4], op2[4];
```

Similar to ladd but computes the integer quotient op1 / op2. If op2 is zero, zero is computed as the result.

```
lmod(r, op1, op2)
char r[4];
char op1[4], op2[4];
```

Similar to ladd but computes op1 mod op2. If op2 is zero, zero is computed as the result.

```
lcomp(op1,op2)
char op1[4], op2[4];
```

Compares longs op1 and op2, and returns one of (the ordinary integers) 1, 0, -1, depending on whether (op1 > op2), (op1 == op2), or (op1 < op2), respectively.

```
lassign(dest,source)
char source[4],dest[4];
```

Assigns long source to long dest, and returns pointer to dest.

```
ltou(l)
char l[4];
```

Converts long l to unsigned (by truncation).

```
utol(l,u)
char l[4];
unsigned u;
```

Stores the long representation of unsigned u into l and returns l.

## 1.2 Implementation Details

Most of the work in the routines above is done by a single 8080 assembly-language function called long, the source for which is found in the file LONG.CSM (available from the C User's Group). The remainder of the package resides in LONG.C. Note that most of the primitives described above simply call long, passing it a function code (that tells it what operation is to be performed) together with the arguments to be manipulated.

The file LONG.CRL contains the compiled functions given in LONG.C, and DEFF2.CRL contains the workhorse function long.

## VITA

William H. Lieber was born on 22 September 1950 in Canton, Ohio. He graduated from Canton McKinley High School in 1968 and attended the University of Akron, Akron, Ohio, from which he received the degree of Bachelor of Science in Electrical Engineering in June 1973. Upon graduation, he entered the Air Force on active duty in December 1973, and received his commission from Officer Training School in April 1974. He served as a scheme engineer at HQ Southern Communications Area Tinker AFB Oklahoma until October 1978. He then served as communications operations officer at the 2006 Communications Group Incirlik Common Defense Installation Turkey until November 1980. He then served as tactical communication equipment acquisition officer for HQ Air Force Communications Command, Scott AFB Illinois, until entering the School of Engineering, Air Force Institute of Technology, in June 1982.

Permanent address: 205 Weber Dr.

O'Fallon Illinois 62269

VITA

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

## REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION UNCLASSIFIED			1b. RESTRICTIVE MARKINGS			
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.			
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE						
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/EE/84D-71			5. MONITORING ORGANIZATION REPORT NUMBER(S)			
6a. NAME OF PERFORMING ORGANIZATION School of Engineering		6b. OFFICE SYMBOL (If applicable) AFIT/EN	7a. NAME OF MONITORING ORGANIZATION			
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright Patterson AFB, Ohio 45433			7b. ADDRESS (City, State and ZIP Code)			
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER			
8c. ADDRESS (City, State and ZIP Code)			10. SOURCE OF FUNDING NOS.			
11. TITLE (Include Security Classification) See Box 19			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
12. PERSONAL AUTHOR(S) William H. Lieber, B.S., Capt, USAF						
13a. TYPE OF REPORT MS Thesis		13b. TIME COVERED FROM _____ TO _____		14. DATE OF REPORT (Yr., Mo., Day) 1984 December		15. PAGE COUNT 255
16. SUPPLEMENTARY NOTATION						
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)			
FIELD	GROUP	SUB. GR.	Analog-to-Digital, Digital-to-Analog, Direct Memory Access, Data Storage, Computer Graphics			
09	02					
19. ABSTRACT (Continue on reverse if necessary and identify by block number)						
Title: DEVELOPMENT OF A DEDICATED SPEECH WORK STATION						
Thesis Chairman: Major Larry R. Kizer						
<div style="text-align: right;"><i>Approved for public release: LAW AFR 190-17.</i> <i>YAN E. WOLAWER</i> 1 May 85 Dean for Research and Professional Development Air Force Institute of Technology (AFIT) Wright Patterson AFB OH 45433</div>						
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS <input type="checkbox"/>			21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED			
22a. NAME OF RESPONSIBLE INDIVIDUAL Major Larry R. Kizer		22b. TELEPHONE NUMBER (Include Area Code) 513-255-3576		22c. OFFICE SYMBOL AFIT/EN		

## 19. ABSTRACT

As a result of the hardware and software developed under this thesis, the AFIT Speech Lab's Cromemco S-100 bus microcomputer system can be configured as a dedicated stand alone speech work station. Hardware is now developed which provides an extended memory capability for storage of analog-to-digital sampled analog speech. Data storage is via a direct memory access (DMA) capability. The hardware also supports providing an analog output from previously stored data samples via a digital-to-analog capability. Software is developed which controls the analog input to be sampled and the sampling rate to be used. The software also allows the sampled data to be graphically displayed 500 samples at a time on a video display screen or to be placed in or returned from more permanent storage on a magnetic disk. The detailed analysis, development, and fabrication of the hardware and software is also contained in the thesis.

**END**

**FILMED**

**8-85**

**DTIC**